改訂3版

FM TOWNS テクニカルデータブック

千葉憲昭 著

アスキー出版局

改訂3版

FM TOWNS テクニカルデータブック

千葉憲昭 著

アスキー出版局

商　標

386，386SX，486，486SX は，米国 Intel 社の商標です．

MS-DOS は，米国 Microsoft 社の登録商標です．

386 | DOS-Extender は，米国 Phar Lap Software 社の商標です．

その他の名称等は一般に各開発メーカーの商標です．

はじめに

1981年のFM8に始まる富士通のFMシリーズのパソコンは，翌年ホビー系(FM7)とビジネス系(FM11)に分化しました．とりわけFM7は大ヒットし，FMシリーズの大衆化に貢献しました．その後ホビー系は3.5インチFDを搭載してFM77(1984年)へと発展し，更にAV(オーディオ＆ビジュアル)時代の幕開けと共にFM77AV(1985年)へと変貌をとげました．

これらのホビー系のパソコンは全て8ビット機でしたが，1989年に入り「パソコンが変わる．TOWNSが変える」のフレーズとともに，一挙に32ビットCPUを搭載したFMTOWNSが登場しました．

オーディオもビジュアルも，取り扱うデータ量が膨大なため，従来CPUのパワー不足が指摘されていたのですが，これにより大幅な改善が図られたことはいうまでもありません．その上，音楽CDも再生できるCDドライブを使ったCD-ROM，高速グラフィックを意図した専用ハードの搭載など思い切った仕様は，新しいパソコンライフの到来を予感させるものです．

しかし，残念なことには，以前にも増して飛躍的に密度が濃くなったハードウエアについて，これまで充分に紹介した資料がなく，TOWNSを活用したくても出来なかったのが実態でした．

そこで，マニアやシステムハウス，ソフトウェアハウス必携のユーザ向けマニュアルとして本書が企画された訳ですが，執筆にあたって筆者はひとつの目標を設定しました．

それは，コンピュータの分野では，従来のマニュアルが仕様書の範囲を充分に脱しきれず，ユーザにとってわかりにくいものになりがちだったので，この際徹底的にこの壁を突き破ろうということでした．FMシリーズの節目にあたって，ハードウェアが刷新されたことは，まさしくこのような試みを実践する大きなチャンスであると考えたのです．

例えば，これまでのハードウェアのマニュアルでは，基本的な説明抜きでいきなり設計された結果のみが記述されることが殆どでしたが，本書ではハードウエアの説明は基礎から解説しています．システムハウスなどの専門家でさえ，レパートリ以外のデバイスについては素人と同じになってしまうという実態からも，基礎的な説明を重視した訳です．とりわけFMTOWNSでは，80386やCD-ROMなど新しいデバイスが数多く使われているので，なおさらというべきでしょう．

もうひとつの本書の特長は，匿名の解析チームのご協力により，メーカが公開していないアドレスマップ，ビットマップを掲載したことです．これらの資料は，システムハウスやソフトウェアハウスのみならず，マニアにとっても不可欠なものだけに，万難を排して掲載に漕ぎつけました．従って，これらについてはメーカの保証範囲外のため，後継機種で予告なく変更されることがありますからご了解下さい．

また本書の第II部では，TownsOS上で利用できるBIOSを解説しました．一般に，個々のレジスタなどの設定によってハードウェアを駆使することは難しくても，BIOSを使えば簡単です．

なお，初代機以降のハードウェアの更新による追加機能については，巻末の付録でまとめて解説しています．また，BIOSについては現時点の最新バージョンの解説になっています．したがって，このBIOSの機能の中には，旧バージョンのBIOSでは使えないものも含まれているので注意してください．

本書を手掛かりとして，FMTOWNSの性能をフルに引き出したソフトやボードの開発が促進されれば，著者としてこれに勝る喜びはありません．

最後に，本書の企画を快諾していただき編集の労をとって頂いたアスキー出版局第三書籍編集部の皆様，側面から協力していただいた解析チームの皆様に心からお礼を申し上げます．

1994年5月　著者　千葉憲昭

本書を利用される場合の御注意

- 本書の第I部では，FMTOWNS(モデル1，モデル2)のハードウェアについて解説しています．なお，新しく追加された機種の仕様変更点については，付録G～Oで解説しています．

- 本書の第II部では，FMTOWNSで利用できる最新のBIOS(FMTOWNS独自のBIOSとFMR-50と互換のBIOSの両方)について解説しています．

- FMTOWNSでは，モデルチェンジによりハードウエアの仕様が変更される場合を考慮して，BIOSレベルで互換性をとることになっています．このため，一般に流通することを目的としてプログラムを作成する場合には，I/Oデバイスを直接アクセスすることはできるだけ避け，BIOSまたは，C言語などを使って行うことが推奨されています．

- 本書の内容については可能な限り万全を期していますが，本書に記述されている事項は，特別な条件下では説明どおりに機能しない場合も考えられます．ハードウェア，ソフトウェアの作成にあたっては，十分に注意して下さい．

- 本書では，386を80386，486を80486と表現しています．

目　　次

第3章　CPU近傍のデバイス　55

第4章　表示システム　97

第5章　オーディオシステム　169

第6章　CD-ROMドライブ　217

第II部 FMTOWNSのBIOS

第1章 BIOSの概要 279

第2章 グラフィックBIOS 291

第3章 スプライトBIOS 365

第4章 マウスBIOS 377

第5章 フォントBIOS 397

第6章 サウンドBIOS 401

第18章　MIDIマネージャBIOS　605

付　録

付録A　各種コネクタの仕様とピン配置　649

付録B　サンプルプログラム　668

BIOSファンクション目次

グラフィックBIOS　20H

スプライトBIOS　60H

マウスBIOS　40H

フォントBIOS　A0H

サウンドBIOS　80H

サウンドBIOS拡張機能　80H

CD-ROM BIOS INT 93H

キーボードBIOS INT 90H

ディスクBIOS　INT 93H

プリンタBIOS　INT 94H

カレンダ時計BIOS　INT 96H

タイマ管理BIOS　INT 97H

時計管理BIOS　INT 98H

RS-232CBIOS　INT 9BH

ブザーBIOS　INT 9EH

割り込み管理BIOS　INT AEH

サービスルーチン　INT AFH

拡張サービスルーチン　INT 8EH

システム情報BIOS 1C0H

音源割り込み管理BIOS 1A0H

MIDIマネージャBIOS　COH

第 I 部

FM TOWNSのハードウェア

ハードウェアの概要

80386CPU の基礎知識

CPU 近傍のデバイス

表示システム

オーディオシステム

CD-ROM ドライブ

各種のデバイス

第 1 章

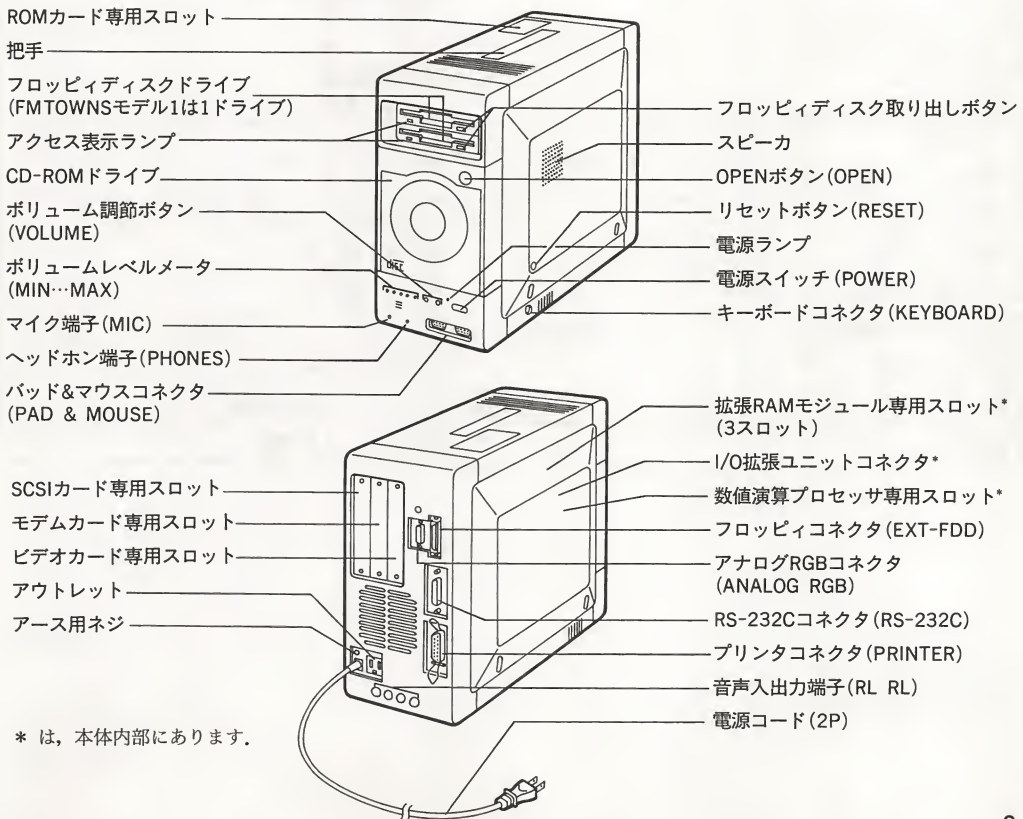
ハードウェアの概要

この章では，FMTOWNS 本体と付属機器の外観と仕様，およびメモリマップとI/Oインタフェースの概要について解説します．

各部の詳細については，次章以降を参照してください．

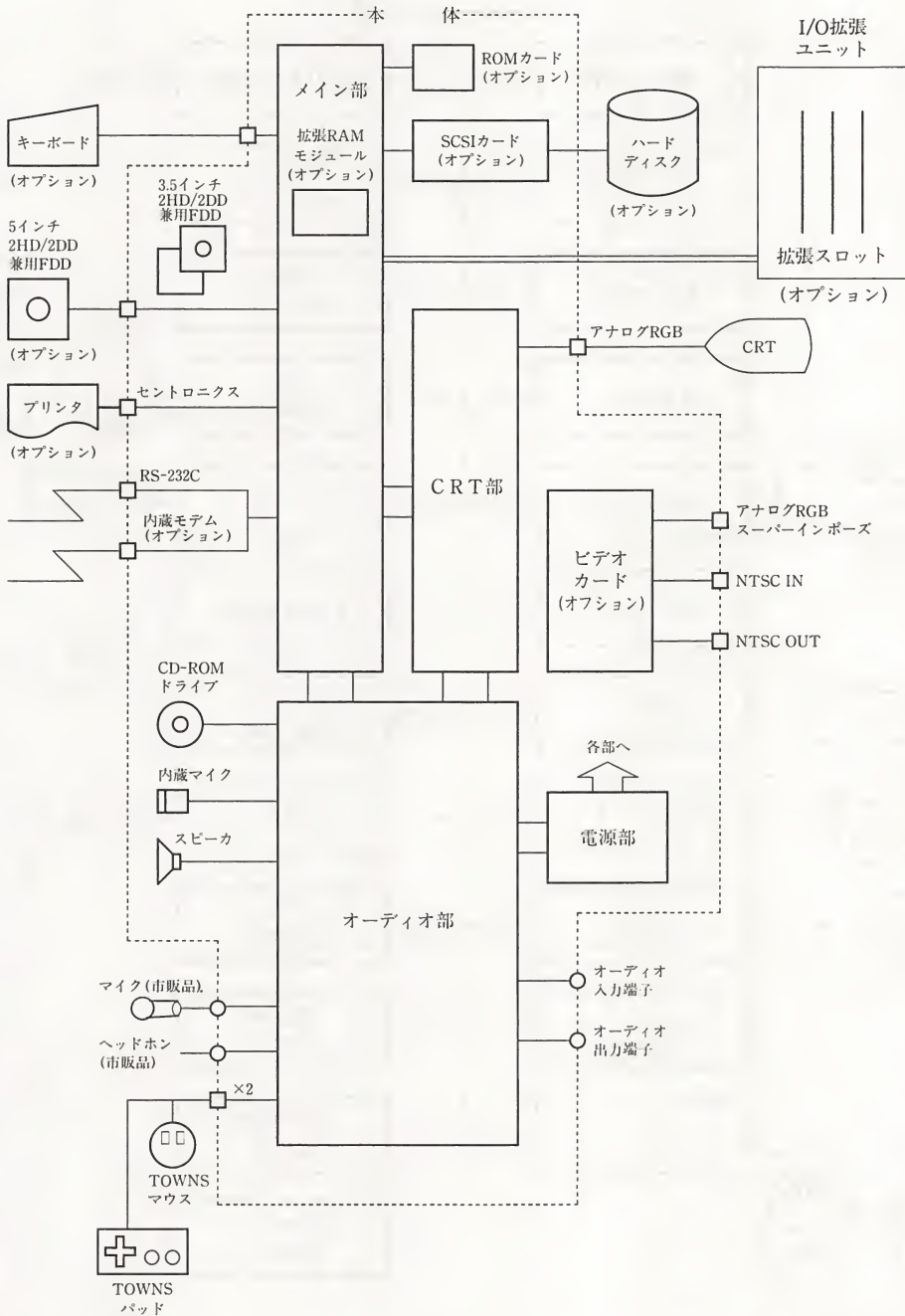
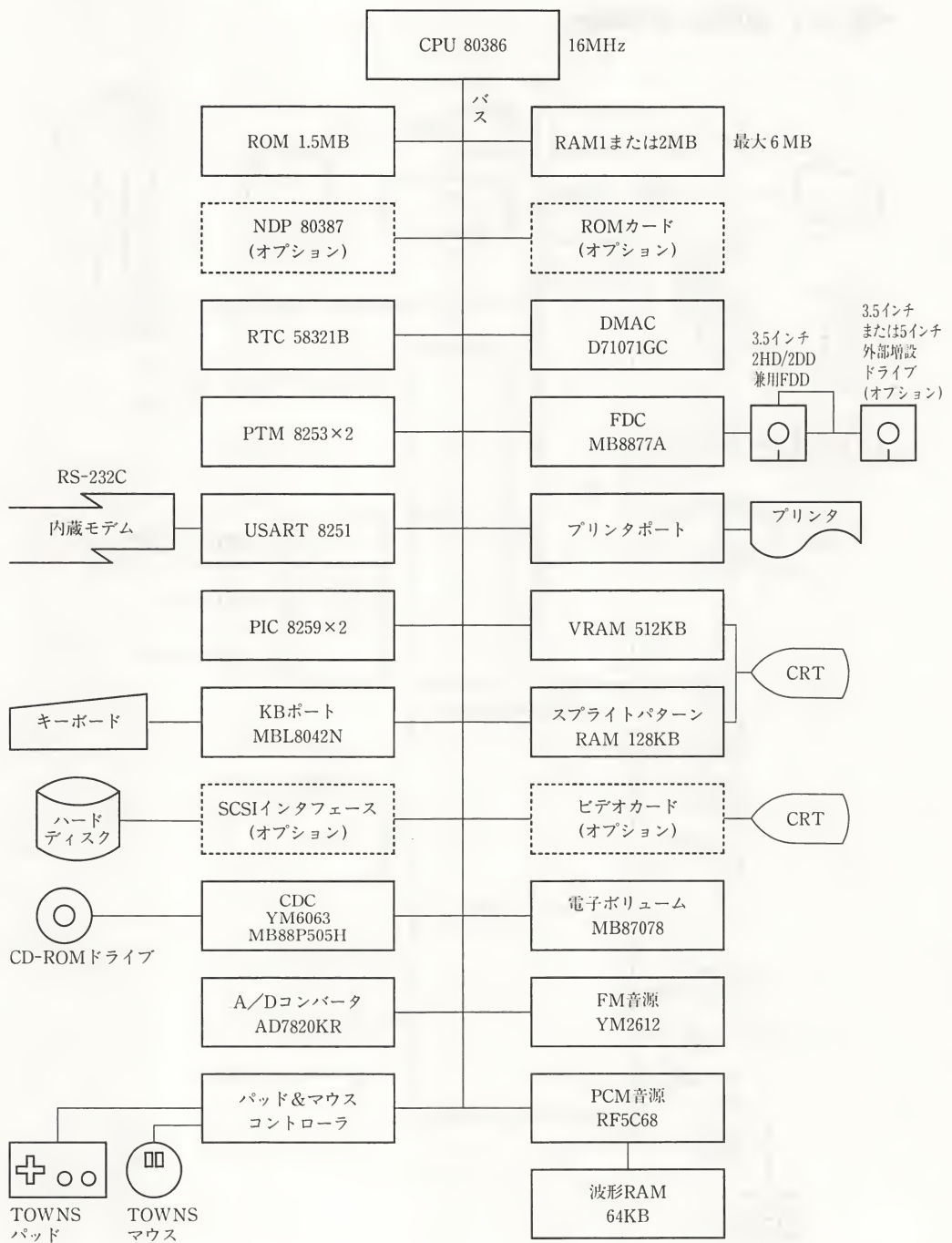
1.1 FMTOWNS の外観と仕様

FMTOWNS 本体の外観と各部の名称を図 I-1-1に，仕様を表 I-1-1に示します．

▼図 I-1-1　FMTOWNS の外観と各部の名称

▼表Ⅰ-1-1　FMTOWNSの仕様

項　目		仕　様	
		モ デ ル 1	モ デ ル 2
CPU 数値演算プロセッサ		80386(16MHz) オプション(80387)	
システムROM OS-ROM ROMカードスロット		256KB 512KB 1スロット	
メインRAM VRAM スプライトRAM		1MB(32ビット) 512KB(32ビット) 128KB(16ビット)	2MB(32ビット) 512KB(32ビット) 128KB(16ビット)
グラフィックス		640×480ピクセル，256/1677万色1面または16/4096色2面 320×240ピクセル，32768色2面 360×240ピクセル，32768色2面など	
漢字ROM 辞書ROM CMOS-RAM		JIS第1，2水準　256KB 512KB 8KB　バッテリバックアップ	
PCM用波形RAM		64KB	
音　源		OPLL(FM音源6声) PCM 8声	
内蔵モデム		オプション	
時計機能		バッテリバックアップ	
インタフェース	CRT FD TOWNSマウス TOWNSパッド RS-232C セントロニクス	アナログRGB内蔵 内　蔵 内　蔵 内　蔵 1回線内蔵 1個内蔵	
インタフェース	SCSI	オプション	
ビデオカード用スロット		1スロット	
キーボード		オプション(親指シフト/JIS)	
TOWNSマウス		標 準 添 付	
TOWNSパッド		標 準 添 付	
補助記憶	3.5インチFD	2HD×1	2HD×2
補助記憶	CD-ROM	1ドライブ内蔵	
補助記憶	ハードディスク	SCSIインタフェースで外部接続	
拡張コネクタ		I/O拡張ユニットを接続	

本体の機器構成を図Ⅰ-1-2に，ブロック図を図Ⅰ-1-3に示します．各種コントローラについては，３章以降で詳細に説明します．

▼図Ⅰ-1-2　FMTOWNS の機器構成

▼図I-1-3 FMTOWNSのブロック図

注)サウンドシステムの詳細なブロック図は,第5章を参照してください。

● CPU

FMTOWNS のCPUには，AV機能を強化するため，大量のデータを高速に処理することができる16MHz の80386が採用されています．

また，オプションの80387数値演算プロセッサを本体に装着すると，数値計算を高速に処理できます．80387は，本体内部の数値演算プロセッサ専用スロットに装着します．

● RAM

出荷時に，ユーザープログラムで利用できるメインRAMとして，モデル1は1MB，モデル2は2MBを搭載しています．その外に，VRAMとして512KB，スプライト用のRAMが128KB，PCM音源の波形格納用のRAM64KBがあります．

また，オプションの拡張RAMモジュール(増設メモリ)を，本体内部の拡張RAMモジュール専用スロットに取り付けると，6MBまでメモリを拡張することができます．

● ROM

システム用のメモリが256KB，OS-ROMが512KB，内蔵されています．その外，漢字ROM256KB，辞書ROM512KB，バッテリバックアップされたCMOS-RAM8KBが内蔵されています．また，本体上部のROMスロットに，立ち上げ時に自動起動するプログラムを書き込んだROMカードを挿入することができます．

●キーボード

キーボードは，オプションでJISキーボードと親指シフトキーボードがあり，それぞれ，テンキーがないものとテンキー付きのものを選択することができます．キーボードインタフェースは，シリアルインタフェース仕様で，FMR-50と互換性があります．

● TOWNSパッド／TOWNSマウス

入力装置として，パッドとインテリジェントマウスが標準装備されています．これまでのFMRシリーズのものと異なる仕様のものであるため，TOWNSパッド，TOWNSマウスと呼んでいます．

●フロッピィディスクドライブ(内蔵／増設)

3.5インチ2HD/2DD兼用タイプのものが，内蔵されています．モデル2では2台内蔵しています．モデル1では1台内蔵していますが，オプションの内蔵マイクロFDドライブを取り付けることにより，内蔵ドライブを2台にできます．

さらに外付けで，5インチの2HD/2DD兼用タイプのFDDユニットを1台接続できます．

●CD-ROMドライブ

読み込み専用の外部補助記憶装置として使われます．CD-ROMは540MBの容量があります．通常のオーディオ用のCDプレーヤとしても使用でき，この場合には，音声はCRTオーディオ部を経由して外部に出力されます．

●CRT／アナログRGB出力

高速スプライト，32768色同時表示，16777216色(以後，本書では1677万色と略記する)から256色選択などの高機能なグラフィック表示システムに対応して，ディスプレイの表示方式はアナログRGB方式となっています．

●オーディオ関係

音源として，PCM音源(ステレオ同時8音)，FM音源(ステレオ同時6音)があります．また，種々の入出力端子を備えており，電子楽器やAV機器を接続することができます．音源の信号と入力した信号をソフトウェアで音量調整し，ミキシングして出力することができます．

内蔵マイクロフォン

本体にマイクが内蔵されており，モノラルで音声を入力できます．

マイク端子

外部マイクを接続するマイク端子です．ステレオマイクも使用できますが，入力経路は1チャネル分のみのため入力音声は左右が混合されたモノラルとなります．

内蔵スピーカ

モニタ用としてスピーカ1個が内蔵されています．再生される音は左右の音声が混合されたものとなります．

ヘッドホン端子

左右2チャンネルの音声出力が出ており，ヘッドホンを接続するとステレオで聞くことができます．

音声入出力端子

アンプやカセットデッキ，ビデオデッキ，電子楽器などを接続することができます．

●セントロニクスインタフェース(プリンタインタフェース)

主にプリンタを接続する，パソコンの標準的なインタフェースです．このポートは，8ビットのパラレル(並列)インタフェースで，出力専用です．

●RS-232C インタフェース

パソコンのインタフェースとして標準的な RS-232C ポートを内蔵しています．RS-232C ポートはシリアルインタフェースで，双方向の通信が可能です．

FMTOWNS には，この外に専用のモデムカードを挿入できるスロットがあり，ここにも，RS-232C ポートの信号線が出ています．同時に使用できるのはどちらか一方のみです．

●スロット

FMTOWNS の背面には，モデムカード，ビデオカード，SCSI カードを装着するための，専用スロットがあります．また，上部には ROM カードを挿入する専用スロットがあります．

SCSI カード

SCSI インタフェースのハードディスクユニットなどが接続できるカードです．

モデムカード

専用スロットに装着すると内蔵のモデムとなり，電話回線を通じた信号のやり取りが可能になります．

ビデオカード

ビデオ入出力端子を備えており，ビデオカメラ，VTR などが接続できるようになり，ビデオデジタイズ，テロップなどさまざまな機能が利用できます．

●I/O 拡張ユニット

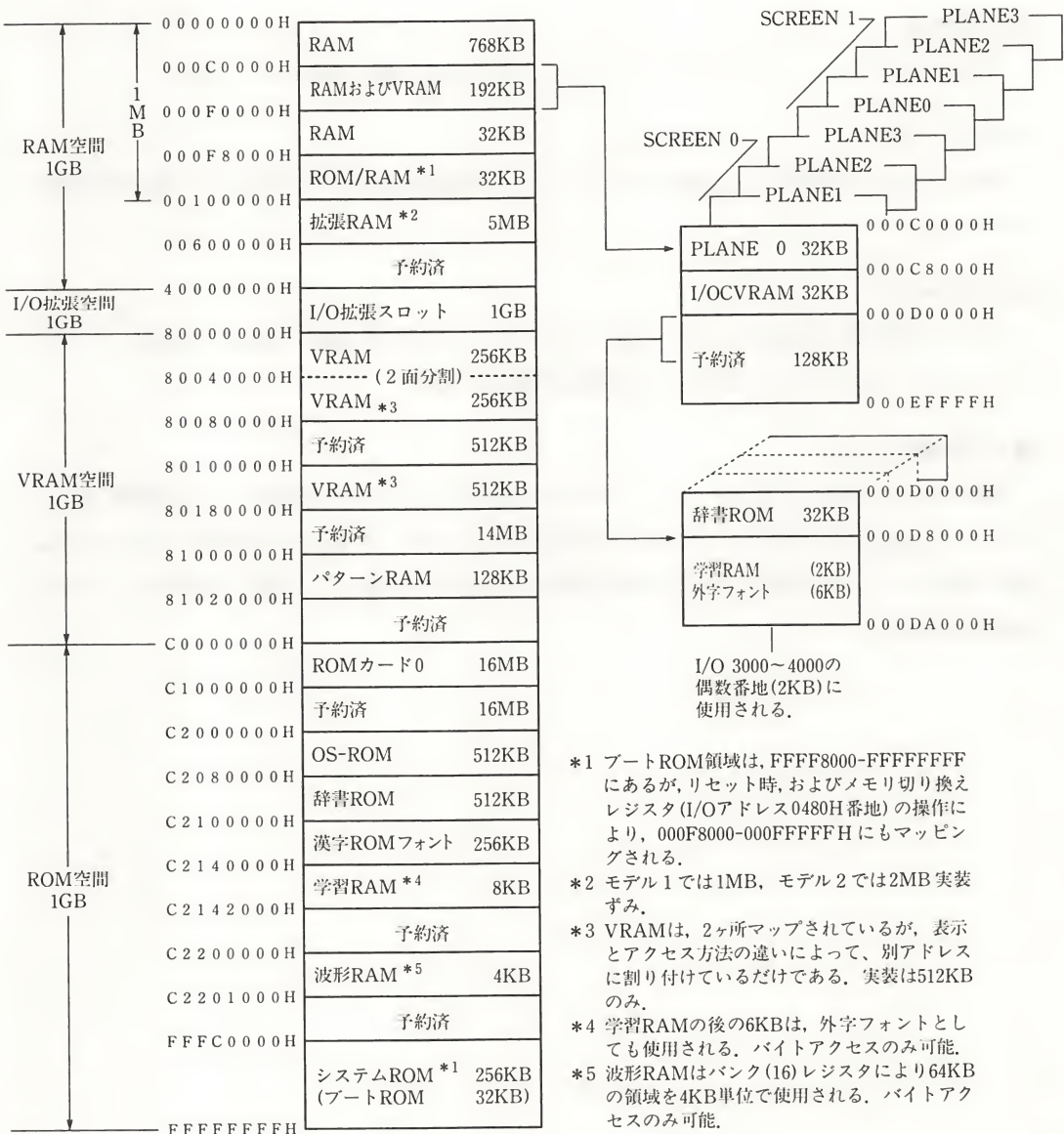
拡張ユニットには，拡張スロットが3スロット用意されており，拡張カードを装着することによって，FMTOWNS のシステムをさらに拡張することができます．I/O 拡張ユニットには，RS-232C カード(RS-232C ポートが本体付属のものだけでは足りない場合)，MIDI カードなどが接続できます．

1.2　メモリマップとI/Oアドレス概要

ここでは，FMTOWNSのメモリマップとI/Oアドレスの一覧を示します．

1.2.1　メモリマップ

図I-1-4に，メモリマップを示します．

▼図I-1-4　メモリマップ

1.2.2 I/Oマップ

I/OインタフェースはCPUと入出力装置(CRT，キーボード，CD-ROMなど)とがデータをやりとりする際の仲介の役目をしています．CPUは直接入出力装置に読み書きすることはできず，入出力装置に対してデータの読み書きを行う際には，その入出力装置に割り当てられたI/Oインタフェースを経由して行います．

各種入出力装置は，さまざまな制御用のLSIなどによって構成されています．I/Oインタフェースに書き込みを行うということは，このような制御用のLSIのレジスタなどへの書き込みを行うということを意味します．

CPUにI/Oインタフェースを接続する方式には，メモリマップドI/OとアイソレーテッドI/Oの2種類があります．

●メモリマップドI/O

メモリの中の特定の番地をI/Oアドレスとして使用するもので，CPUに6809を使用している8ビット機(ホビー系FMシリーズ)などで採用されています．

●アイソレーテッドI/O

メモリ以外の別空間にI/Oアドレスを設定しているもので，I/Oとメモリは独立しています．8086，80286，80386などではこの形式を採用しています．

●FMTOWNSのI/O

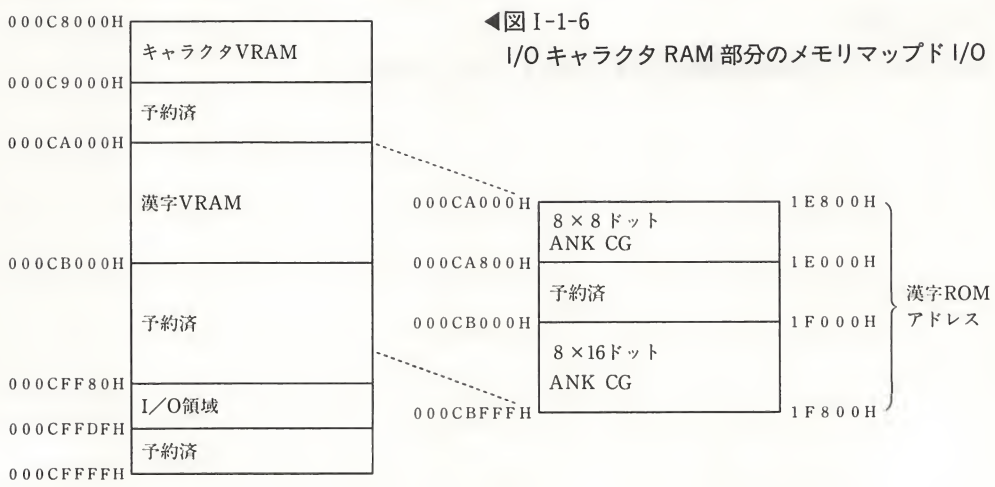
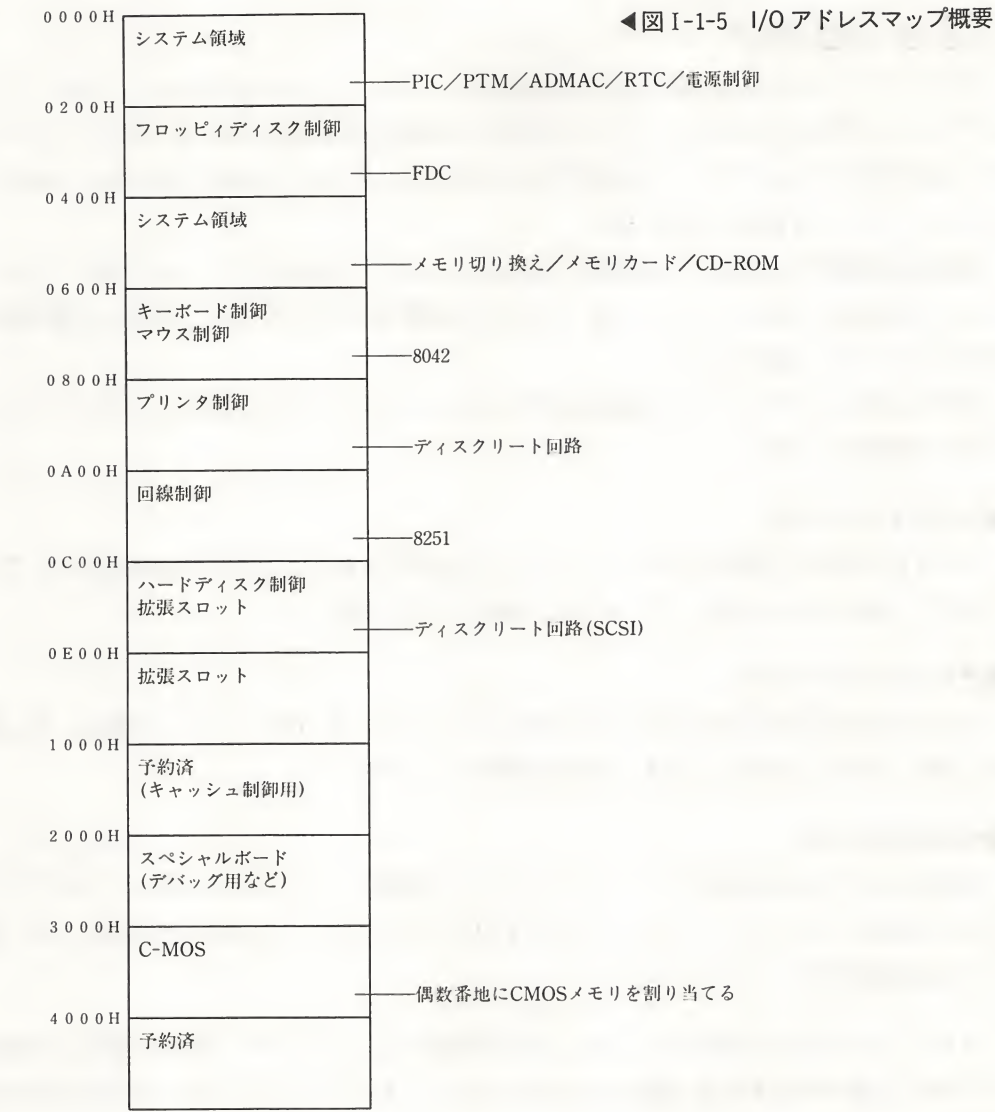
FMTOWNSでは，基本的にアイソレーテッドI/Oを採用していますが，一部にメモリマップドI/Oを採用しています．アイソレーテッドI/Oのアドレスマップを図I-1-5に示します．また，その詳細を表I-1-2に示します．

メモリマップドI/Oの部分としては，IO/CVRAM(I/OキャラクタVRAM部分)の32KB中の000CFF80～000CFFDFの部分があります(図I-1-6)．また，4GBのうち1GB分がI/Oの拡張スロット用に当てられており，デバイスの拡張時には，この空間をI/O用として使用することになります．

メモリマップドI/Oの領域の内容を，表I-1-3に示します．

◀図I-1-5　I/Oアドレスマップ概要

アドレス	領域	内容
0000H	システム領域	PIC／PTM／ADMAC／RTC／電源制御
0200H	フロッピィディスク制御	FDC
0400H	システム領域	メモリ切り換え／メモリカード／CD-ROM
0600H	キーボード制御 マウス制御	8042
0800H	プリンタ制御	ディスクリート回路
0A00H	回線制御	8251
0C00H	ハードディスク制御 拡張スロット	ディスクリート回路(SCSI)
0E00H	拡張スロット	
1000H	予約済 (キャッシュ制御用)	
2000H	スペシャルボード (デバッグ用など)	
3000H	C-MOS	偶数番地にCMOSメモリを割り当てる
4000H	予約済	

◀図I-1-6
I/Oキャラクタ RAM 部分のメモリマップド I/O

▼表 I-1-2　I/O アドレスマップ表(アイソレーテッド I/O)

I/O アドレス	レジスタ名	R/W	ビット構成								Word / Byte	備考	参照
			7	6	5	4	3	2	1	0			
0000	インタラプトリクエストレジスタ　(マスタ側)	R	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0	B	PIC (8259A) マスタ	表 I -3-2
	インタラプトサービスレジスタ　(マスタ側)	R	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0			表 I -3-3
	動作コマンドワード 2 (マスタ側)	W	R	SL	EOI	0	0	L2	L1	0			表 I -3-12
	動作コマンドワード 3 (マスタ側)	W	0	ESMM	SMM	0	1	P	PR	RIS			表 I -3-13
	初期化コマンドワード 1 (マスタ側)	W	A7	A6	A5	1	LTIM	ADI	SNGL	IC4			表 I -3-7
0002	インタラプトマスクレジスタ　(マスタ側)	R	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0			表 I -3-4
	動作コマンドワード 1 (マスタ側)	W	M7	M6	M5	M4	M3	M2	M1	M0			表 I -3-11
	初期化コマンドワード 2 (マスタ側)	W	A15/T7	A14/T6	A13/T5	A12/T4	A11/T3	A10	A9	A8			表 I -3-8
	初期化コマンドワード 3 (マスタ側)	W	S7	S6	S5	S4	S3	S2	S1	S0			表 I -3-9
	初期化コマンドワード 4 (マスタ側)	W	0	0	0	SFNM	BUF	M/S	AEOI	μPM			表 I -3-10
0010	インタラプトリクエストレジスタ　(スレーブ側)	R	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0	B	PIC (8259A) スレーブ	表 I -3-2
	インタラプトサービスレジスタ　(スレーブ側)	R	IS7	IS6	IS5	IS4	IS3	IS2	IS1	IS0			表 I -3-3
	動作コマンドワード 2 (スレーブ側)	W	R	SL	EOI	0	0	L2	L1	L0			表 I -3-12
	動作コマンドワード 3 (スレーブ側)	W	0	ESMM	SMM	0	1	P	PR	RIS			表 I -3-13
	初期化コマンドワード 1 (スレーブ側)	W	A7	A6	A5	1	LTIM	ADI	SNGL	IC4			表 I -3-7
0012	インタラプトマスクレジスタ (スレーブ側)	R	IR15	IR14	IR13	IR12	IR11	IR10	IR9	IR8			表 I -3-4
	動作コマンドワード 1 (スレーブ側)	W	M15	M14	M13	M12	M11	M10	M9	M8			表 I -3-11
	初期化コマンドワード 2 (スレーブ側)	W	A15/T7	A14/T6	A13/T5	A12/T4	A11/T3	A10	A9	A8			表 I -3-8
	初期化コマンドワード 3 (スレーブ側)	W	0	0	0	0	0	ID2	ID1	ID0			表 I -3-9
	初期化コマンドワード 4 (スレーブ側)	W	0	0	0	SFNM	BUF	M/S	AEOI	μPM			表 I -3-10

<table>
<tr><th rowspan="2">I/O
アドレス</th><th rowspan="2">レジスタ名</th><th rowspan="2">R/W</th><th colspan="8">ビット構成</th><th rowspan="2">Word
/
Byte</th><th rowspan="2">備考</th><th rowspan="2">参照</th></tr>
<tr><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0020</td><td>リセット要因レジスタ</td><td>R</td><td colspan="6">不定</td><td>SHUT DOWN</td><td>SOFT</td><td rowspan="2">B</td><td rowspan="2"></td><td>表I-3-34</td></tr>
<tr><td>ソフトリセット，NMIベクタプロテクト，ソフト電源制御</td><td>W</td><td>WR PROT</td><td>POW OFF</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>RST</td><td>表I-3-35</td></tr>
<tr><td>0022</td><td>電源制御レジスタ</td><td>W</td><td>0</td><td>POW OFF</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>B</td><td></td><td>表I-3-36</td></tr>
<tr><td rowspan="2">0030</td><td rowspan="4">CPU 識別レジスタ</td><td rowspan="2">R</td><td colspan="5">MACHINE-ID</td><td colspan="3">CPU-ID</td><td rowspan="4">B/W</td><td rowspan="4"></td><td rowspan="4">表I-3-37</td></tr>
<tr><td>ID7</td><td>ID6</td><td>ID5</td><td>ID4</td><td>ID3</td><td>ID2</td><td>ID1</td><td>ID0</td></tr>
<tr><td rowspan="2">0031</td><td rowspan="2">R</td><td colspan="8">MACHINE-ID</td></tr>
<tr><td>ID15</td><td>ID14</td><td>ID13</td><td>ID12</td><td>ID11</td><td>ID10</td><td>ID9</td><td>ID8</td></tr>
<tr><td rowspan="2">0032</td><td rowspan="2">シリアルROM制御レジスタ</td><td>R</td><td rowspan="2">ID RESET</td><td rowspan="2">ID CLK</td><td colspan="5">不　定</td><td>ID DATA</td><td rowspan="2">B</td><td rowspan="2"></td><td rowspan="2">表I-3-38</td></tr>
<tr><td>W</td><td>CS ID</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td rowspan="2">0040</td><td rowspan="6">タイマカウントレジスタ PIT1</td><td rowspan="2">R/W</td><td colspan="8">タイマ#0（インターバルタイマ　307.2kHz）</td><td rowspan="2">B</td><td rowspan="8">PIT1 (8253)</td><td rowspan="6">表I-3-25</td></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td rowspan="2">0042</td><td rowspan="2">R/W</td><td colspan="8">タイマ#1（I/O 制御用　307.2kHz）</td><td rowspan="2">B</td></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td rowspan="2">0044</td><td rowspan="2">R/W</td><td colspan="8">タイマ#2（サウンド　307.2kHz）</td><td rowspan="2">B</td></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>0046</td><td>コントロールレジスタ1 PIT1（#0～#2）</td><td>W</td><td>SC1</td><td>SC0</td><td>RL1</td><td>RL0</td><td>M2</td><td>M1</td><td>M0</td><td>BCD</td><td>B</td><td>表I-3-27</td></tr>
<tr><td rowspan="2">0050</td><td rowspan="6">タイマカウントレジスタ PIT2</td><td rowspan="2">R/W</td><td colspan="8">タイマ#3（予約済）</td><td rowspan="2">B</td><td rowspan="8">PIT2 (8253)</td><td rowspan="6">表I-3-25</td></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td rowspan="2">0052</td><td rowspan="2">R/W</td><td colspan="8">タイマ#4（ボーレートジェネレータ　1.2299kHz）</td><td rowspan="2">B</td></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td rowspan="2">0054</td><td rowspan="2">R/W</td><td colspan="8">タイマ#5（予約済）</td><td rowspan="2">B</td></tr>
<tr><td>D7</td><td>D6</td><td>D5</td><td>D4</td><td>D3</td><td>D2</td><td>D1</td><td>D0</td></tr>
<tr><td>0056</td><td>コントロールレジスタ2 PIT2（#3～#5）</td><td>W</td><td>SC1</td><td>SC0</td><td>RL1</td><td>RL0</td><td>M2</td><td>M1</td><td>M0</td><td>BCD</td><td>B</td><td>表I-3-27</td></tr>
<tr><td rowspan="2">0060</td><td>割り込み要因レジスタ</td><td>R</td><td colspan="3">不定</td><td>SOUND</td><td>TM1 MSK</td><td>TM0 MSK</td><td>TM OUT1</td><td>TM OUT0</td><td>B</td><td rowspan="2"></td><td>表I-3-29</td></tr>
<tr><td>割り込み制御レジスタ</td><td>W</td><td>TM0 CLR</td><td>0</td><td>0</td><td>0</td><td>0</td><td>SOUND</td><td>TM1 MSK</td><td>TM0 MSK</td><td>B</td><td>表I-3-28</td></tr>
<tr><td rowspan="2">0070</td><td rowspan="2">RTC データレジスタ</td><td>R</td><td>READY</td><td colspan="3">不定</td><td rowspan="2">D3</td><td rowspan="2">D2</td><td rowspan="2">D1</td><td rowspan="2">D0</td><td>B</td><td rowspan="3">RTC インタフェース</td><td rowspan="2">表I-3-32</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>B</td></tr>
<tr><td>0080</td><td>RTC コマンドレジスタ</td><td>W</td><td>CHIP SELECT</td><td>0</td><td>0</td><td>0</td><td>0</td><td>READ</td><td>WRITE</td><td>ADRS WRITE</td><td>B</td><td>表I-3-33</td></tr>
<tr><td>00A0</td><td>イニシャライズレジスタ</td><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>16B</td><td>RES</td><td>B</td><td>DMAC</td><td>表I-3-14</td></tr>
</table>

I/O アドレス	レジスタ名	R/W	ビット構成								Word / Byte	備考	参照
			7	6	5	4	3	2	1	0			
00A1	チャネルレジスタ	R	不定			BASE	SELCH				B	DMAC	表I-3-15
							SEL3	SEL2	SEL1	SEL0			
		W	0	0	0	0	0	BASE	SELCH		B		
									SL1	SL0			
00A2	カウントレジスタ(下位)	R/W	C7	C6	C5	C4	C3	C2	C1	C0	B/W		表I-3-16
00A3	カウントレジスタ(上位)	R/W	C15	C14	C13	C12	C11	C10	C9	C8			
00A4	アドレスレジスタ(下位)	R/W	A7	A6	A5	A4	A3	A2	A1	A0	B/W		表I-3-17
00A5	アドレスレジスタ(中位)	R/W	A15	A14	A13	A12	A11	A10	A9	A8			
00A6	アドレスレジスタ(上位)	R/W	A23	A22	A21	A20	A19	A18	A17	A16			
00A7	アドレスレジスタ(最上位)	R/W	A31	A30	A29	A28	A27	A26	A25	A24			
00A8	デバイスコントロールレジスタ	R/W	AKL	RQL	EXW	ROT	CMP	DDMA	AHLD	MTM	W		表I-3-18
00A9		R	不定						WEV	BHLD	W		
		W	0	0	0	0	0	0					
00AA	モードコントロールレジスタ	R	TMODE		ADIR	AUTI	TDIR		不定	W/B	B		表I-3-19
			MD1	MD0			DIR1	DIR2					
		W	TMODE				TDIR		0				
			MD1	MD0			DIR1	DIR2					
00AB	ステータスレジスタ	R	REQUEST				TERMINAL COUNT				B		表I-3-20
			RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0			
00AC	テンポラリレジスタ(下位)	R	T7	T6	T5	T4	T3	T2	T1	T0	B		表I-3-21
00AD	テンポラリレジスタ(上位)	R	T15	T14	T13	T12	T11	T10	T9	T8			
00AE	リクエストレジスタ	R	不定				SRQ3	SRQ2	SRQ1	SRQ0	B		表I-3-22
		W	0	0	0	0							
00AF	マスクレジスタ	R	不定				M3	M2	M1	M0	B		表I-3-23
		W	0	0	0	0							
00B0	イニシャライズレジスタ	W	0	0	0	0	0	0	16B	RES	B	拡張DMAC	表I-3-14
00B1	チャネルレジスタ	R	不定			BASE	SELCH				B		表I-3-15
							SEL3	SEL2	SEL1	SEL0			
		W	0	0	0	0	0	BASE	SELCH		B		
									SL1	SL0			
00B2	カウントレジスタ(下位)	R/W	C7	C6	C5	C4	C3	C2	C1	C0	B/W		表I-3-16
00B3	カウントレジスタ(上位)	R/W	C15	C14	C13	C12	C11	C10	C9	C8			

I/O アドレス	レジスタ名	R/W	ビット構成								Word/Byte	備考	参照
			7	6	5	4	3	2	1	0			
00B4	アドレスレジスタ(下位)	R/W	A7	A6	A5	A4	A3	A2	A1	A0	B/W	拡張DMAC	表I-3-17
00B5	アドレスレジスタ(中位)	R/W	A15	A14	A13	A12	A11	A10	A9	A8			
00B6	アドレスレジスタ(上位)	R/W	A23	A22	A21	A20	A19	A18	A17	A16			
00B7	アドレスレジスタ(最上位)	R/W	A31	A30	A29	A28	A27	A26	A25	A24			
00B8	デバイスコントロールレジスタ	R/W	AKL	RQL	EXW	ROT	CMP	DDMA	AHLD	MTM	W		表I-3-18
00B9		R	不定						WEV	BHLD	W		
		W	0	0	0	0	0	0					
00BA	モードコントロールレジスタ	R	TMODE		ADIR	AUTI	TDIR		不定	W/B	B		表I-3-19
			MD1	MD0			DIR1	DIR2					
		W	TMODE		ADIR	AUTI	TDIR		0				
			MD1	MD0			DIR1	DIR2					
00BB	ステータスレジスタ	R	REQUEST				TERMINAL COUNT				B		表I-3-20
			RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0			
00BC	テンポラリレジスタ(下位)	R	T7	T6	T5	T4	T3	T2	T1	T 0	B		表I-3-21
00BD	テンポラリレジスタ(上位)	R	T15	T14	T13	T12	T11	T10	T9	T 8			
00BE	リクエストレジスタ	R	不定				SRQ3	SRQ2	SRQ1	SRQ0	B		表I-3-22
		W	0	0	0	0							
00BF	マスクレジスタ	R	不定				M3	M2	M1	M0	B		表I-3-23
		W	0	0	0	0							
0200	ステータスレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0	B	FDC (8877A)	表I-7-24
	コマンドレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0			表I-7-23
0202	トラックレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0	B		表I-7-25
0204	セクタレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0	B		
0206	データレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0	B		
0208	ドライブステータスレジスタ	R	不定					3.5FDD	FREADY	1	B	FD インタフェース	表I-7-26
	ドライブコントロールレジスタ	W	0	0	CLK SEL	MOTOR	0	HD1 SEL	DDEN	IRQ MSK			表I-7-27
020C	ドライブセレクトレジスタ	W	0	HISPD	0	INUSE	DSL3	DSL2	DSL1	DSL0	B		表I-7-28

I/O アドレス	レジスタ名	R/W	ビット構成								Word / Byte	備考	参照
			7	6	5	4	3	2	1	0			
020E	ドライブスイッチレジスタ	R/W	0	0	0	0	0	0	0	DRV CHG	B	FDインタフェース	表I-7-29
0400	システムステータスレジスタ	R	不定							解像度	B		表I-3-39
0402	予約済												
0404	システムステータスレジスタ	R	MAIN MEM	不定							B	VRAM, RAM切り換えレジスタ	表I-3-39
		W		0	0	0	0	0	0	0			
0406 ～ 043E	予約済												
0440	アドレスレジスタ	W	0	0	0	RA4	RA3	RA2	RA1	RA0	B	CRTC	表I-4-24
0442	データレジスタ(下位)	W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	W/B		
0443	データレジスタ(上位)	W	RD15	RD14	RD13	RD12	RD11	RD10	RD9	RD8	W/B		
0448	アドレスレジスタ	W	0	0	0	0	0	0	RA1	RA0	B	ビデオ出力コントローラ	表I-4-37
044A	データレジスタ	W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	B		
044C	デジタルパレットモディファイフラグレジスタ	R	DPMD	不定					SPD0	PAGE	B	FMR互換	表I-4-38
0450	アドレスレジスタ	R	不定					RA2	RA1	RA0	B	スプライトコントローラ	表I-4-16
		W	0	0	0	0	0						
0452	データレジスタ	R/W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	B		
0458	アドレスレジスタ	R	不定						RA1	RA0	B	VRAMアクセスコントローラ	表I-4-5
		W	0	0	0	0	0	0					
045A	データレジスタ(下位)	R/W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0	W/B		
045B	データレジスタ(上位)	R/W	RD15	RD14	RD13	RD12	RD11	RD10	RD9	RD8			
0480	メモリ切り換えレジスタ	R	不定						RAM	辞書ROM	B		表I-3-40
		W	0	0	0	0	0	0					
0484	辞書ROM	R	不定				DBK3	DBK2	DBK1	DBK0	B		表I-3-41
		W	0	0	0	0							
0486 ～ 0488	予約済												
048A	メモリカードステータス	R	CHANGE	不定	RED	YELLOW	(RDY)	CD-0	CD-1	WP	B		表I-3-42
048C ～ 04BE	予約済												

I/O アドレス	レジスタ名	R/W	ビット構成 7	6	5	4	3	2	1	0	Word / Byte	備考	参照
04C0	マスタステータスレジスタ	R	SIRQ	DEI	STSF	DTSF	不定		SRQ	DRY	B	CDC	表I-6-2
	マスタコントロールレジスタ	W	SMIC	DEIC	0	0	0	SRST	SMIM	DEIM			表I-6-1
04C2	ステータスレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0	B		表I-6-5
	コマンドレジスタ	W	TYPE	IRQ	STATUS	COMMAND CODE							表I-6-3
04C4	データレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0	B		表I-6-7
	パラメータレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0			表I-6-4
04C6	転送制御レジスタ	W	0	0	0	DTS	STS	0	0	0	B		表I-6-6
04CC	CDサブコードステータスレジスタ	R	不　定						OVER RUN	SUBC DATR	B		表I-6-8
04CD	CDサブコードデータレジスタ	R	SUBC P-DATA	SUBC Q-DATA	SUBC R-DATA	SUBC S-DATA	SUBC T-DATA	SUBC U-DATA	SUBC V-DATA	SUBC W-DATA	B		表I-6-9
04D0	パッド1入力レジスタ	R	不定	COM	TRIG2	TRIG1	RIGHT1	LEFT	BACK	FWD	B	パッドインタフェース	表I-7-9
04D2	パッド2入力レジスタ	R	不定	COM	TRIG2	TRIG1	RIGHT1	LEFT	BACK	FWD	B		
04D5	FM・PCMミュートレジスタ	R	不　定						FM MUTE	PCM MUTE	B	ミュートレジスタ	表I-5-43
		W	0	0	0	0	0	0					
04D6	パッド出力レジスタ	W	0	0	JOY2 COM	JOY1 COM	JOY2 TRIG2	JOY2 TRIG1	JOY1 TROG2	JOY1 TRIG1	B	パッドインタフェース	表I-7-10
04D8	ステータスレジスタ	R	BUSY	不定					FLAG B	FLAG A	B	FM音源(YM2612)	表I-5-21
	アドレスレジスタ0	W	タイマ，チャネル1〜3　アドレス										
			A7	A6	A5	A4	A3	A2	A1	A0			
04DA	データレジスタ0	W	タイマ，チャネル1〜3　ライトデータ								B		
			D7	D6	D5	D4	D3	D2	D1	D0			
04DC	アドレスレジスタ1	W	チャネル4〜6　アドレス								B		
			A7	A6	A5	A4	A3	A2	A1	A0			
04DE	データレジスタ1	W	チャネル4〜6　ライトデータ								B		
			D7	D6	D5	D4	D3	D2	D1	D0			
04E0	ボリューム1　DATAレジスタ	R	不定		D5	D4	D3	D2	D1	D0	B	電子ボリュームレジスタ	表I-5-1
		W	0	0									
04E1	ボリューム1　COMレジスタ	R	不定			C32	C0	EN	CH1	CH0	B		
		W	0	0	0								
04E2	ボリューム2 DATAレジスタ	R	不定		D5	D4	D3	D2	D1	D0	B		
		W	0	0									
04E3	ボリューム2 COMレジスタ	R	不定			C32	C0	EN	CH1	CH0	B		
		W	0	0	0								

I/O アドレス	レジスタ名	R/W	ビット構成 7	6	5	4	3	2	1	0	Word / Byte	備考	参照
04E7	AD サンプリングデータレジスタ	R	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0	B	AD コンバータ	表 I -5-4
04E8	AD サンプリングフラグレジスタ	R	不定							サンプリングフラグ	B		表 I -5-5
		W	0	0	0	0	0	0	0				
04E9	INT13 割り込み要因レジスタ	R	不定				PCM	不定		FM	B		表 I -5-17
04EA	PCM 割り込みマスクレジスタ	R/W	M7	M6	M5	M4	M3	M2	M1	M0	B	PCM 音源	表 I -5-14
04EB	PCM 割り込みレジスタ	R	IF7	IF6	IF5	IF4	IF3	IF2	IF1	IF0	B		表 I -5-15
04EC	オーディオレジスタ	R	LOFF	MUTE	不定						B		表 I -5-42
		W			1	1	1	1	1	1			
04F0	ENV データレジスタ	W	ENV D7	D6	D5	D4	D3	D2	D1	D0	B	PCM 音源 (RF5C68)	表 I -5-10
04F1	PAN データレジスタ	W	RIGHT PAN3	PAN2	PAN1	PAN0	LEFT PAN3	PAN2	PAN1	PAN0	B		表 I -5-11
04F2	FDL データレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0	B		表 I -5-8
04F3	FDH データレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0	B		
04F4	LSL データレジスタ	W	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0	B		表 I -5-9
04F5	LSH データレジスタ	W	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	B		
04F6	ST データレジスタ	W	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8	B		表 I -5-7
04F7	コントロールレジスタ	W	ON/OFF	MOD	0	0	WB3 / 0	WB2 / CB2	WB1 / CB1	WB0 / CB0	B		表 I -5-12
04F8	チャネル ON/OFF レジスタ	W	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1	B		表 I -5-13
0500 ～ 0502	予約済												
05C0	NMI マスクレジスタ	R	不定				BNMI	不定			B		表 I -3-43
		W	0	0	0	0		0	0	0			
05C2	NMI ステータスレジスタ	R	不定				BNMI	不定			B		表 I -3-44
05C4 ～ 05C6	予約済												
05C8	TVRAM 書き込みレジスタ	R	MD	不定							B		表 I -3-45
05CA	VSYNC 割り込み原因クリアレジスタ	W	WRITE → クリア								B		表 I -3-46

I/O アドレス	レジスタ名	R/W	ビット構成								Word / Byte	備考	参照
			7	6	5	4	3	2	1	0			
05CC 〜 05D0	予約済												
05E0 〜 05FE	予約済												
0600	キーボードデータレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0	B	キーボード インタ フェース	表I-7-4
	8042データレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0	B		表I-7-6
0602	ステータスレジスタ	R	ST7	ST6	ST5	ST4	F1	F0	IBF	OBF	B		表I-7-5
	コマンドレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0	B		表I-7-2
0604	割り込み要因フラグレジスタ	R	不定						NMI	KBINT	B		表I-7-7
	割り込み制御レジスタ	W	0	0	0	0	0	0	0	KBMSK	B		表I-7-8
0800	ステータスレジスタ1	R	BUSY	PE	FUSE	THSN	POW	ACK	FAULT	PREADY	B	プリンタ インタ フェース	表I-7-14
	データレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0	B		表I-7-13
0802	ステータスレジスタ2	R	不定				SLCT	RINF3	RINF2	RINF1	B		表I-7-15
	コントロールレジスタ	W	0	0	0	0	0	EX PRM	IN PRM	DMA	B		表I-7-17
0804	割り込み制御レジスタ	W	0	0	0	0	0	0	ACK MSK	FLT MSK	B		表I-7-16
0A00	受信データレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0	B	USART (8251)	表I-7-39
	送信データレジスタ	W											
0A02	ステータスレジスタ1	R	DSR	SYNDET /BD	FE	OE	PE	TxE	RxRDY	TxRDY	B		表I-7-38
	同期モードレジスタ	W	SCS	ESD	EP	PEN	L2	L1	0	0	B		表I-7-36
	非同期モードレジスタ	W	S2	S1	EP	PEN	L2	L1	B2	B1	B		表I-7-35
	コマンドレジスタ	W	EH	IR	RTS	ERR RST	SBRK	RxEN	DTR	TxE	B		表I-7-37
0A04	ステータスレジスタ2	R	不定				DSR	CD	CS	CI	B	RS-232C インタ フェース	表I-7-40
0A06	割り込み要因レジスタ	R	不定					CI	CS	RSINT	B		表I-7-41
0A08	割り込み制御/クロック切り換えレジスタ	W	TxC	RxC	EXT DTR	CI	CS	SYN DET	RxRDY	TxRDY	B		表I-7-42

I/O アドレス	レジスタ名	R/W	ビット構成								Word / Byte	備考	参照
			7	6	5	4	3	2	1	0			
0A0A	モデム制御レジスタ	R	ENBL	MODEM	MODINS	不定					B	RS-232C インタフェース	表I-7-43
		W			0	0	0	0	0	0			
0C30	データレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0	B	SCSI インタフェース	表I-7-33
0C32	ステータスレジスタ	R	REQ	I/O	MSG	C/D	BUSY	不定	INT	PERR	B		表I-7-32
	コントロールレジスタ	W	WEN	IMSK	0	ATN	0	SEL	DMAE	RST	B		表I-7-31
FD90	アナログパレットコード	R/W	P7	P6	P5	P4	P3	P2	P1	P0	B	アナログパレットレジスタ	表I-4-6
FD92	青色のパレットデータ	R/W	BL7	BL6	BL5	BL4	BL3	BL2	BL1	BL0	B		
FD94	赤色のパレットデータ	R/W	RL7	RL6	RL5	RL4	RL3	RL2	RL1	RL0	B		
FD96	緑色のパレットデータ	R/W	GL7	GL6	GL5	GL4	GL3	GL2	GL1	GL0	B		
FD98	パレットデータ0	R	不定				C3	C2	C1	C0	B	デジタルパレットレジスタ	表I-4-40
		W	0	0	0	0							
FD99	パレットデータ1	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FD9A	パレットデータ2	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FD9B	パレットデータ3	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FD9C	パレットデータ4	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FD9D	パレットデータ5	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FD9E	パレットデータ6	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FD9F	パレットデータ7	R	不定				C3	C2	C1	C0	B		
		W	0	0	0	0							
FDA0	SUBステータスレジスタ	R	不定						HSYNC	VSYNC	B	FMR互換	表I-4-45
	CRT出力コントロールレジスタ	W	0	0	0	0	画面レイア0 COLOR	画面レイア0 GREEN	画面レイア1 COLOR	画面レイア1 GREEN	B		表I-4-39

▼表I-1-3 I/Oアドレスマップ表(メモリマップドI/O)

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0	Word / Byte	備考	参照
000C FF80	MIXレジスタ	R	不定		CURSOR LSB	不定	WIDTH	不定			B	ダミー	表I-4-42
		W	0	0		0		0	0	0			
000C FF81	グラフィックVRAM 更新モードレジスタ	R	READOUT CNTRL		不定		RAM SELECT BIT				B	FMR 互換	表I-4-43
			RC2	RC1			RAM4	RAM3	RAM2	RAM1			
		W	READOUT CNTRL		0	0	RAM SELECT BIT						
			RC2	RC1			RAM4	RAM3	RAM2	RAM1			
000C FF82	グラフィックVRAM ディスプレイモード レジスタ	R	不定								B	FMR 互換	表I-4-41
		W	0	1	RAM	PAGE SELECT		RAM SELECT BIT					
						PS2	0	RAM3	RAM2	RAM1			
000C FF83	グラフィックVRAM ページセレクトレジスタ	R	不定			PAGE SELECT		不定			B	FMR 互換	表I-4-44
						PS2	0						
		W	0	0	0	PAGE SELECT		0	0	0			
						PS2	0						
000C FF84	FIRQレジスタ	R	0	不定							B	予備	表I-3-47
000C FF86	STATUSレジスタ	R	HSYSNC	不定		1	不定	VSYNC	不定		B	FMR 互換	表I-4-46
000C FF88 ～ 000C FF8F	予約済												
000C FF94	漢字CG アクセスレジスタ	R	L2CG	不定							B		表I-3-48
		W	KC15	KC14	KC13	KC12	KC11	KC10	KC9	KC8	B		
000C FF95		W	KC7	KC6	KC5	KC4	KC3	KC2	KC1	KC0	B		
000C FF96		R/W	D15	D14	D13	D12	D11	D10	D9	D8	B		
000C FF97		R/W	D7	D6	D5	D4	D3	D2	D1	D0	B		
000C FF98	ブザー制御レジスタ	R/W	Read → ON								B		表I-3-49
			Write → OFF										
000C FF99	漢字VRAMレジスタ	W	0	0	0	0	0	0	0	ANKCG	B		表I-3-50
000C FFA0	論理演算レジスタ	R	ESTART								B		表I-3-51
000C FFA1 ～ 000C FFBB	予約済												

第 2 章

80386CPUの基礎知識

この章では，FMTOWNSに採用されている80386CPUの概要を説明します．80386は，広大なメモリ空間を扱える高速なCPUです．また，マルチタスクOSを強く意識して設計されており，従来のCPUと比べ，OSを作成する際に有用な機能が大幅に拡張されています．

章の前半では，ソフトウェアで参照できるレジスタについて，その種類や特徴などを説明します．

後半では，80386を特徴づけているアドレス生成機構と記憶保護機構などを扱います．この部分は，OSに関係する内容が多いので，OS上で動作するアプリケーションプログラムなどを作成するにはほとんど無縁ですが，FMTOWNSの基本OSであるTownsOSでどのような処理が行われているかを理解する手助けにはなることと思います．

なお，80386CPUは非常に強力な(機能の豊富な)CPUであり，この紙幅では全体を詳細に述べることはできません．詳しい知識が必要な場合は，80386CPUの解説書を参照してください．

2.1 80386CPUの特徴

この節では，80386CPUの特徴を，いくつかの側面に集約して述べます．

●32ビットの汎用レジスタとデータバス

80386CPUは，32ビットの汎用レジスタとデータバスを持っています．ここでいうビット数とは，同時にアクセスできるデータ長を指すもので，4バイト分のデータを一度に読み書きできることを意味します．これにより，従来は何度にも分けて行っていた処理が1度で済むようになり，それだけ処理は高速になっています．

●従来のCPUの機能を継承しながら機能を拡張

80386の前身となった，8086，80186，80286などのCPUは，いずれも16ビットCPUで，データバスのサイズも80386の半分しかありません．これらのCPUは，基本的に1MBのアドレス空間を64KB単位に分割して管理するアーキテクチャになっており，これがグラフィックス表示

の際の高速描画や，多くのメモリを必要とするオーディオデータを扱う際の障害になっていました．80386は，このような障害を取り除き，かつ従来のCPUの機能も内包しています．

●広大なアドレス空間をサポート

80386では，4GB(ギガバイト)の連続したアドレス空間(プロテクトモード時)が実現されており，前述の障害を解決しています．さらに，仮想アドレスという概念も採用し，64TB(テラバイト)のアドレス空間も扱えるようになりました(仮想アドレスは80286から採用されていますが，80286の仮想アドレス空間は1GBです)．

ただし，この場合の仮想アドレスは，アドレスバス上のアドレス空間まで拡張するものではありません．複数のプログラムが同一のアドレス空間を使用する際に，これらを分散させるように働く点に注意が必要です．この概念については，この章の後半で詳しく説明します．

●マルチタスク，マルチユーザーのOSを考えたCPU

80386は，複数のプログラムを同時に実行するための機能を備えており，時分割によって，マルチタスクを実行するTSS(タイムシェアリングシステム)にも対応できるようになっています．

マルチタスクは，見かけ上複数プログラムが同時に実行されているように見えますが，実際はCPUが個々のプログラムを順次切り換えて実行しているにすぎません．80386は，タスクを高速に切り換える機能をハードウェアで持っています．

●メモリの保護をするハードウェア

マルチタスクでは，複数のプログラムが並行して実行できればそれでよいということではなく，個々のプログラムが独立して実行されなければなりません．言い換えると，あるプログラムが暴走するなどの不都合な動作をしても，他のプログラムに影響が及ぶようでは困ります．

このために，80386では，タスク間でお互いの領域を犯すことのないように，記憶保護を行っています(タスク間保護)．また，タスクを4段階の階層(特権レベル)に分け，例えばOS部分は特権レベル0(最高の優先度)，その他は優先度に応じて特権レベル1～3に差別化することで，優先度の低い側のタスクからは，優先度の高い側のタスクのメモリをアクセスすることを禁止しています(リング型保護)．また，リング型保護では，特権レベル0でなければ使用できない命令(特権命令)もあります．上位側のプログラムは下位の側を管理しなければならないので，下位の側で不都合があっても上位側に影響しないようにするわけです．

●処理の高速化

CPUの内部機能は，いくつかのユニットに分かれており，命令を実行する段階の1つを担当しています．各ユニットは，処理の結果を隣のユニットに渡すとともに，次の命令の処理にかかります．このようなパイプライン処理により，各ユニットが休みなく動作することができ，処理速度が速くなります．

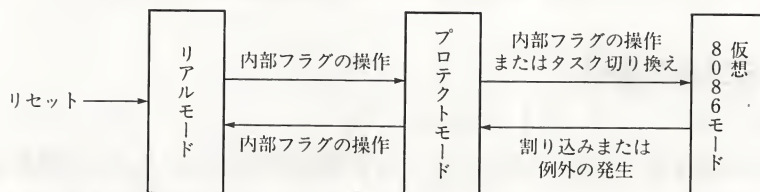
また，キャッシュは，アクセスしたメモリブロックの内容をCPU内部に保存するもので，メモリのアクセス回数を低減する働きがあります．

●デバッグ機能

デバッグに必要な機能を，ハードウェア化して内蔵しています．シングルステップ実行と，4ヶ所までのブレークポイントを設定することができ，ブレークポイントは，命令だけではなく，データにも設定できます．

2.2　3種類の動作モード

80386は，最も基本的な「プロテクトモード(ネイティブモード)」，8086などとの互換性を確保するための「リアルモード」と「仮想8086モード」の3種類の動作モードがあります．これらのモードの関係を図I-2-1に示します．

▼図I-2-1　80386の3つのモードの切り換え

●リアルモード

80386は，リセットすると，最初にリアルモードになります．このモードは，8086と互換性があります．ただし，仮想記憶，記憶保護，マルチタスク制御は行われません．

FMTOWNSの電源をONに(またはリセット)したときに起動されるTownsOSは，まずこのモードで起動し，次にプロテクトモードに移ります．

●プロテクトモード

プロテクトモードは，80386のすべての機能を発揮するモードです．リアルモードからプロテクトモードに切り換えるには，コントロールレジスタ(後述)のフラグを操作します．通常，この操作はOSが行います．プロテクトモードでは，仮想記憶，記憶保護，マルチタスク制御がサポートされます．このモードは，80286のプロテクトモードも包括しているので，80286のプログラムも動かすことができます．

TownsOSでの，ユーザーのプログラムは，このモードで働きます．

●仮想8086モード

プロテクトモードでは，複数のタスクを実行することができ，その1つとして8086のプログラムを動かすことができます．

仮想8086モードは，プロテクトモード下で8086のプログラムを動かすためのモードです．プロテクトモードの他のタスク(80286，80386のプログラム)と混在することができます．また，複数の仮想8086モードのタスクを実行することもできます．このモードは，プロテクトモードの保護優先度の最下位(特権レベル3，後述)に置かれており，8086，80186用のソフトウェアは，修正なしで動作します．

なお，TownsOSは基本的にシングルタスクのOSなので，このモードを使用していません．

2.3　レジスタ

80386内部のレジスタは，8086／88，80186／188，80286との互換性を保つために，これらのCPUのレジスタを拡張した形で設計されています．

この節では，これらのレジスタの構成と各レジスタの機能を説明します．

2.3.1　レジスタの構成

80386のすべてのレジスタを，図I-2-2に示します．

図のあみかけの部分は，現在広く利用されている8086CPUのレジスタと共通する部分です．多くのレジスタが，16ビットから32ビットに拡張されていることが分かります．また，「システムレジスタ」と呼ばれる一群のレジスタが追加されています．システムレジスタは，通常は，OSがシステムを管理するために使用するレジスタです．

アプリケーションプログラムに関連するレジスタは，汎用レジスタ，セグメントレジスタ，インストラクションポインタ，フラグレジスタです．

以下で，各レジスタの種別ごとに説明を行います．

2.3.2　汎用レジスタ/セグメントレジスタ/インストラクションポインタ/フラグレジスタ

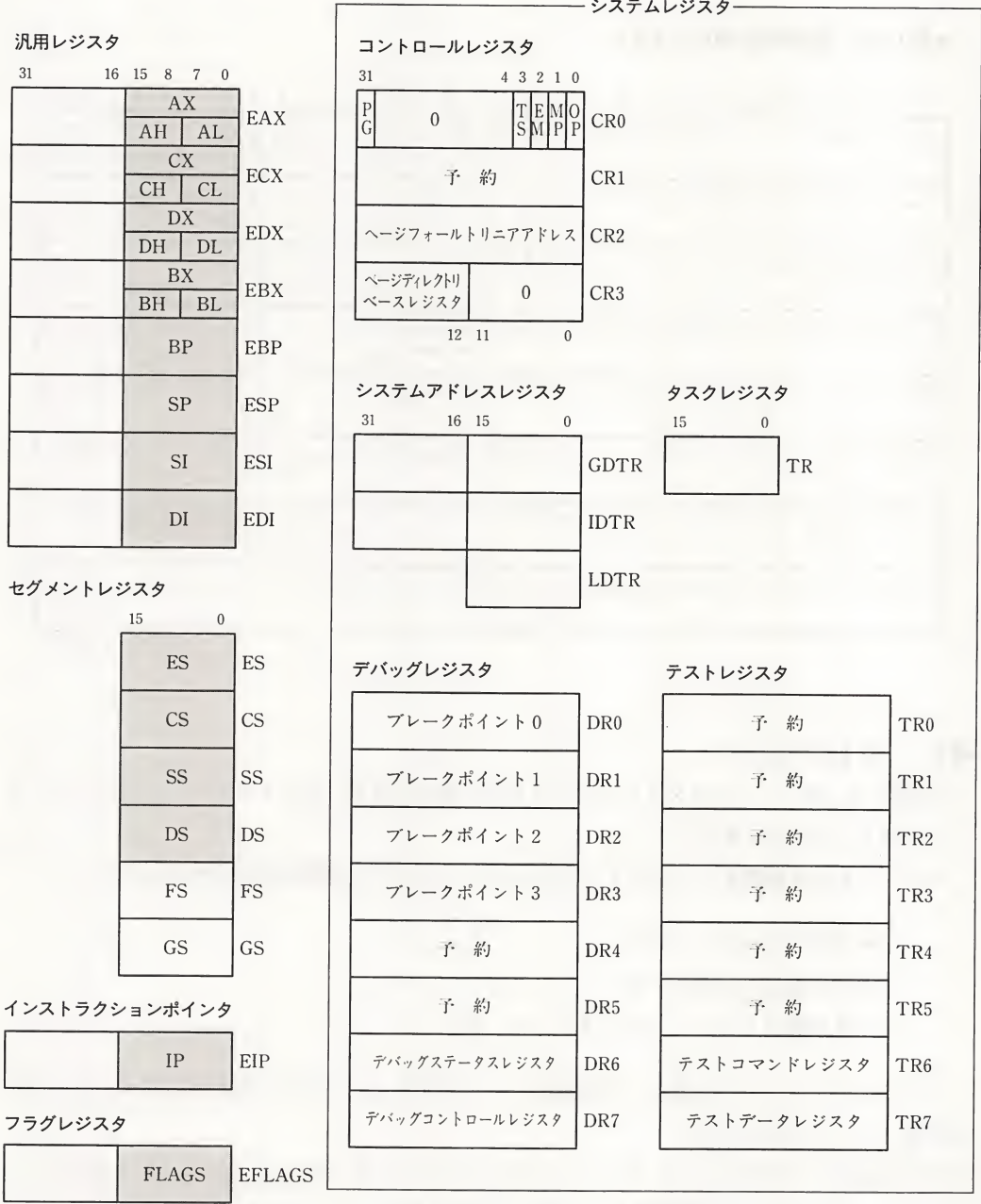
●汎用レジスタ

80386には，32ビットの汎用レジスタが8本あります．汎用レジスタのフォーマットを図I-2-3に示します．

8086，80286などのCPUには，AX，CX，DX，BX，SP，BP，SI，DIの各16ビットの汎用レジスタがありました．80386は，これらに対応して，EAX，ECX，EDX，EBX，ESP，EBP，ESI，EDIの32ビットの汎用レジスタを持っています．各レジスタ名が〝E〟で始まって

いることに注目してください．

これらのレジスタを32ビットで扱うときは，〝E××〟のように呼びます．先頭の〝E〟を除いて，２文字のレジスタ名で扱うと(AX，DIのように)，従来どおりの16ビットレジスタとしてアクセスすることができます．

▼図Ⅰ-2-2　80386の全レジスタ構成

さらに，AX，CX，DX，BXの各レジスタは，8ビット単位でもアクセスできます．この場合は，例えば，AXレジスタの上位8ビットをAH，下位8ビットをALのようにアクセスします．

ところで，従来のCPUでは〝汎用レジスタ〟といっても，実際には個別に役割が決められていて，専用レジスタの域を出ませんでした．これに対して，80386の拡張された32ビット汎用レジスタは文字どおり，四則演算にもアドレス修飾用のメモリポインタとしても使用できます．

▼図I-2-3　80386の汎用レジスタ

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

31～16	15～8	7～0
E A X		
	AX	
	AH	AL
E C X		
	CX	
	CH	CL
E D X		
	DX	
	DH	DL
E B X		
	BX	
	BH	BL
E S P		
	S P	
E B P		
	B P	
E S I		
	S I	
E D I		
	D I	

(32ビット×8本)

●セグメントレジスタ

80386には，16ビットのセグメントレジスタが6本あります．セグメントレジスタのフォーマットを図I-2-4に示します．

セグメントは3種類あり，使用するセグメントレジスタとの関係は次のとおりです．

コードセグメント………CS
スタックセグメント……SS
データセグメント………DS，ES，FS，GS

スタックセグメントの内容は，ある種のデータですから，これも一種のデータセグメントとして考えることができます．

データセグメント用の，FS，GSのセグメントレジスタは，80386で追加されたものです．これにより，80386では，4種類のデータ領域を同時に扱うことができます．

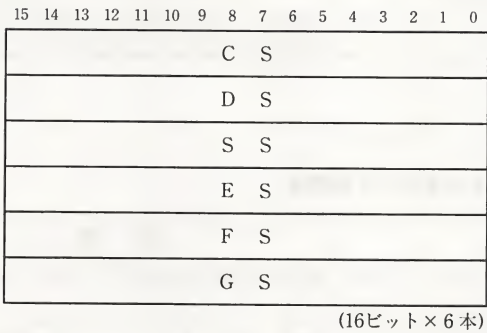
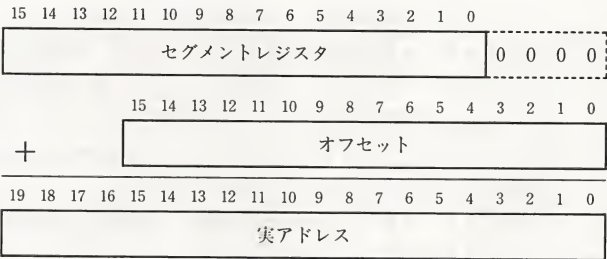
セグメントレジスタの働きは，80386の動作モードによって異なります．

リアルモードと仮想8086モードでは，セグメントレジスタ中の値は16倍(4ビット左シフト)され，命令のオペランドなどで与えられるオフセット値と加算されて，実アドレスを計算するために使用されます．このようすを，図I-2-5に示します．

プロテクトモードでは，セグメントレジスタは，後述の「セグメントディスクリプタテーブル」という，セグメントを管理する表の項目を指すセレクタ値を持ちます．

▼図I-2-4　80386のセグメントレジスタ

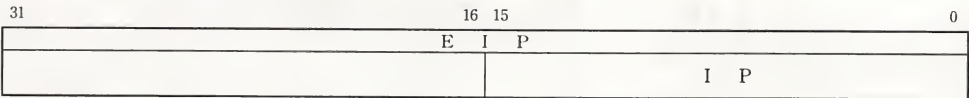
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
C S
D S
S S
E S
F S
G S

(16ビット×6本)

▼図I-2-5　リアルモード，仮想8086モードでのセグメントレジスタの動作

●インストラクションポインタ(命令ポインタ)

インストラクションポインタは，次に実行する命令(インストラクション)のアドレスを指しています．インストラクションポインタのフォーマットを図I-2-6に示します．

命令ポインタの0～15ビットは，汎用レジスタと同様に，従来のCPUに対応するものです．この部分だけを，IPと呼びます

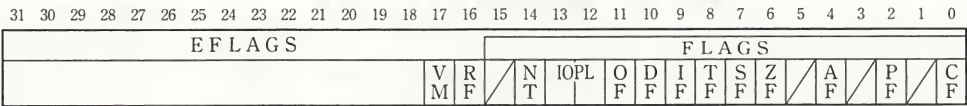
▼図I-2-6　80386のインストラクションポインタ

31 … 16	15 … 0
E I P	
	I P

●フラグレジスタ

フラグレジスタは，演算結果やCPUの状態などを参照／設定するビットの集まりです．フラグレジスタのフォーマットを図I-2-7に，各フラグの意味を，表I-2-1に示します．

フラグレジスタの0～15ビットは，従来のCPUに対応するものです．この部分だけを，FLAGSと呼びます．

▼図I-2-7　80386のフラグレジスタ

31〜18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EFLAGS			FLAGS															
	VM	RF	/	NT	IOPL		OF	DF	IF	TF	SF	ZF	/	AF	/	PF	/	CF

▼表I-2-1　フラグレジスタの各ビットの意味

ビット位置	名前	機　能
0	CF	キャリーフラグ：演算の結果，最上位ビットでキャリーやボローが生じたときにセットされる．
2	PF	パリティフラグ：演算の結果，下位8ビットに偶数個の1があるときにセットされる．
4	AF	補助キャリーフラグ：演算の結果，ビット3でキャリーやボローが生じたときにセットされる．
6	ZF	ゼロフラグ：演算の結果，0になるとセットされる．
7	SF	符号フラグ：演算の結果の最上位ビットを表す．（結果が正なら0，負なら1）
8	TF	シングルステップフラグ：一度セットすると，次の命令を実行するごとにシングルステップ例外を生じさせる．TFは割り込みによってクリアされる．
9	IF	割り込み許可フラグ：セットすると，マスク可能割り込みが生じたとき，CPUの制御は割り込みベクタの示す位置に移行する．
10	DF	方向フラグ：ストリング命令実行時に，自動的に対応するインデックスレジスタの増減を指定する．セットされているときはデクリメントを行い，クリアされているとインクリメントを行う．
11	OF	オーバーフローフラグ：符号付演算でオーバーフローが生じたとき，つまり結果がディスティネーションに入りきらないときにセットされる．
12～13	IOPL	I/O特権レベル：I/Oデバイスに対して階層構造のプロテクトを働かせるとき，そのレベルを設定する．
14	NT	ネステッドタスクフラグ：タスクの下にネストされたタスクがあるときセットされる．
16	RF	レジュームフラグ：命令の実行が終了しているかどうかを示し，ページフォルトにより実行を完了していないときにセットされる．
17	VM	仮想モードフラグ：仮想8086モードに移行するためのフラグ．プロテクトモードまたはページプロテクトモードのときセットすると，仮想8086モードに移行する．セットするにはIRET命令の実行，またはプロテクトモードでのタスク切り換えによる．割り込みや例外によって自動的にクリアされる．

2.3.3 システムレジスタ

●コントロールレジスタ

コントロールレジスタは，CPUの動作モードやページング機構を，OSが制御するために用意されたものです．80386には，32ビットのコントロールレジスタが4本(CR0〜CR3)あります．CR0は動作モードと数値演算プロセッサの種類と接続の有無の制御，CR2とCR3はページングの制御に使用します．CR1は，現在のところ使われていません．

コントロールレジスタのフォーマットを図I-2-8に示します．図中の〝予約済″の部分は，ユーザーは使用できず，仕様も公開されていません．

CR0の各ビットの意味を，表I-2-2に示します．

▼図I-2-8　80386のコントロールレジスタ

(32ビット×4本)

▼表I-2-2　CR0のビットの意味

名称	意　味
PG	ページングのイネーブル/ディセーブルを行う． 1＝ページング機能のイネーブル 0＝ページング機能のディセーブル
TS	タスクスイッチが発生したことを示すビット． 1＝タスクスイッチが発生した 0＝タスクスイッチが発生していない このビットが1のままでESC命令を実行すると，トラップ7が発生する．
EM	ESC命令でトラップを発生するか，プロセッサへ送るかを決める． 1＝トラップを発生(トラップ7) 0＝プロセッサと通信する
MP	TSビットと関連して使われ，MP＝1，TS＝1でESC命令とWAIT命令の両方でトラップ7を発生する．
PE	セグメントベースのプロテクション機能のイネーブル/ディセーブルをする． 1＝イネーブル 0＝ディセーブル(リアルモード)

PG，PEビットは，組み合せによってCPUの動作を規定します．組み合せを，表I-2-3に示します．両ビットが0の場合はリアルモードで(起動時)，両ビットが1の場合に，80386は全能力を発揮します．プロテクトモードで，ページングを使用しないことはできますが(PG＝0，PE＝1)，リアルモードでのページングの使用(PG＝1，PE＝0)はできません．

CR2，CR3は，ページング処理のためのレジスタです．

CR2には，ページング動作の失敗(ページフォールト)時に，原因となった処理ルーチンのリニアアドレスが格納されます．これはページング処理のエラーハンドラで利用できます．

CR3は，ページング処理で使用されるページディレクトリのベースアドレスを保持します．80386では，1ページが4KBに固定されているので，このレジスタの下位12ビットは常に0です．

ページング処理に関しては，「2.5.2　仮想記憶管理と実アドレスへの変換」に解説があります．

▼表I-2-3　PG，PEビットと動作モード

PG	PE	動作モード
0	0	リアルモード
0	1	プロテクトモード(ページングなし)
1	0	使用せず
1	1	プロテクトモード(ページングあり)

●システムアドレスレジスタ

4本のシステムアドレスレジスタは，セグメント処理によるアドレス生成や，タスクの切り換えなどのために使用されるレジスタです．

システムアドレスレジスタのフォーマットを，図I-2-9に示します．

80386は，アドレス生成時のセグメント処理に，ディスクリプタテーブルと呼ばれるデータテーブルを参照します．このテーブルの各項目は「ディスクリプタ」と呼ばれ，各種セグメントの属性，サイズ，先頭アドレスを含んでいます．ディスクリプタやセグメント処理に関しては，「2.5.2　仮想記憶管理と実アドレスへの変換」に解説があります．

図中の「セレクタ」とは，テーブル中の項目(ディスクリプタ)を指す番号です．「ベース」はセグメントの先頭アドレス，「リミット」はセグメントのサイズです．

それぞれのレジスタは，次のような働きをします．

GDTRは，GDT(グローバルディスクリプタテーブル)のベースアドレスとリミットを規定します．GDTは，どのタスクからも参照できるディスクリプタのテーブルです．

LDTRは，LDT(ローカルディスクリプタテーブル)のセレクタ(番号)を保持します．LDTは，タスクごとの固有のディスクリプタテーブルで，他のタスクからは参照できません．

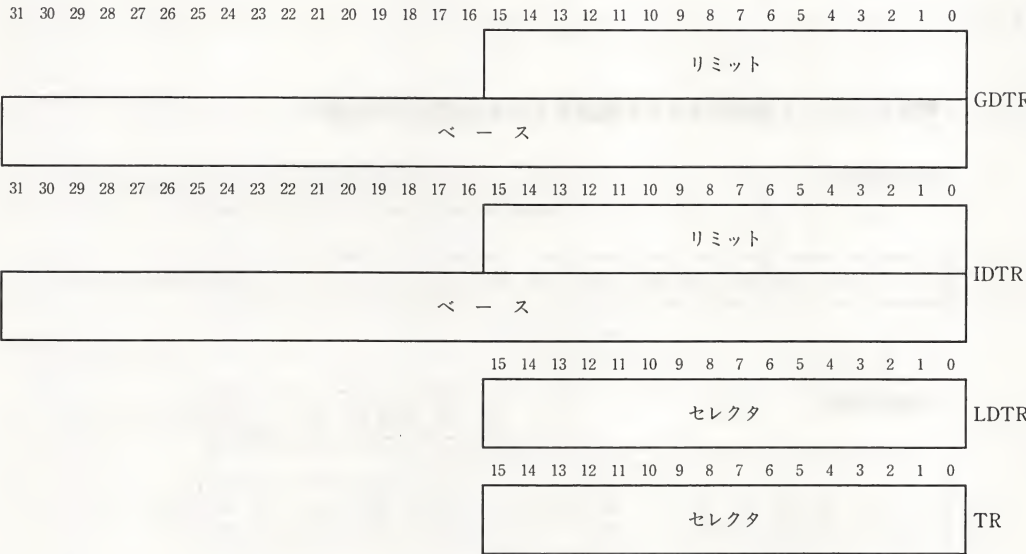
IDTRは，IDT(インタラプトディスクリプタテーブル)のベースアドレスのリミットを規定

します．IDTは，割り込みや例外の発生時に参照されるディスクリプタテーブルです．

TRは，TSS(タスクステートセグメント)へのセレクタを設定するものです．TSSは，タスク切り換え時にレジスタなどの状態を保存し，再度そのタスクに制御がもどったときに復元できるようにするためのものです．

TSSは，各タスクに固有のデータ構造体で，タスクの属性やレジスタの待避領域などからなります．したがって，個々のタスクについて1つずつ存在します．

なお，TSS(タスクステートセグメント)という語は，インテル社の造語です．広く使用されている「タイムシェアリングシステム」の略語ではないので注意してください．

▼図I-2-9　80386のシステムアドレスレジスタ

●デバッグレジスタ

80386は，デバッグ機能をハードウェア化して内蔵しています．デバッグに関するレジスタには，デバッグレジスタが8本(DR0～DR7)あります．ブレークポイントのアドレスは，4ヶ所まで設定でき，DR0～DR3に設定します．

●テストレジスタ

80386は，ページング処理の高速化のためのキャッシュ(TLB)を内蔵していますが，その状態を調べるためのレジスタです．テストレジスタは，7本(TR0～TR7)ありますが，TR0～TR5はインテルによって予約されています．TR6がテストレジスタ，TR7がテストデータレジスタとして使われます．

2.4　2進データをメモリに収容するときの注意点

ここでは，FM-7など6809CPUに慣れた読者のために，80386CPUのデータバス上でのビットの対応について説明します．

6809などのモトローラ系のCPUでは，データバスとレジスタの各ビット値の対応は，完全にストレートですが，インテル系のCPUの場合，8ビット(1バイト)単位の逆順になっています．このようすを，図I-2-10に示します．

これは，インテル社が4004CPU(4ビットのCPU)を開発後，8ビットに拡張するときに，拡張部分を後ろに付け足したのがそのまま習わしになってしまったためと思われます．

このため80386では，32ビットデータは，図I-2-11のような対応づけによって，バスに出力されます．

▼図I-2-10　2進値をメモリに転送するときの対応付けの違い

6089の場合

b7 b6 b5 b4 | b3 b2 b1 b0　8ビットデータ

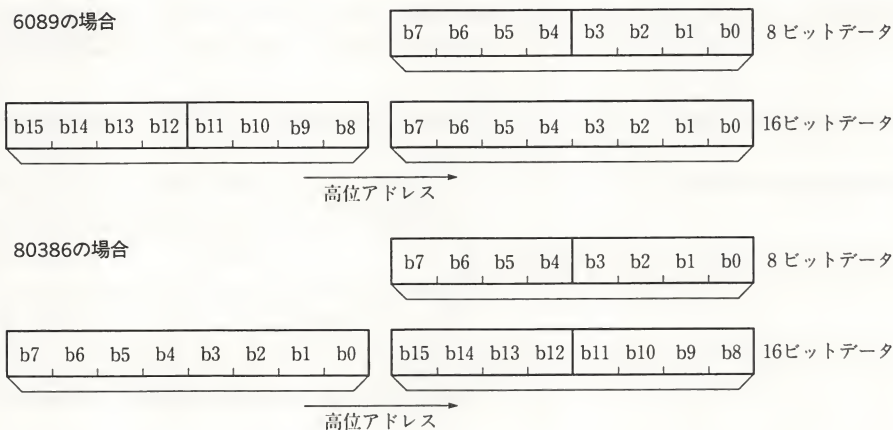
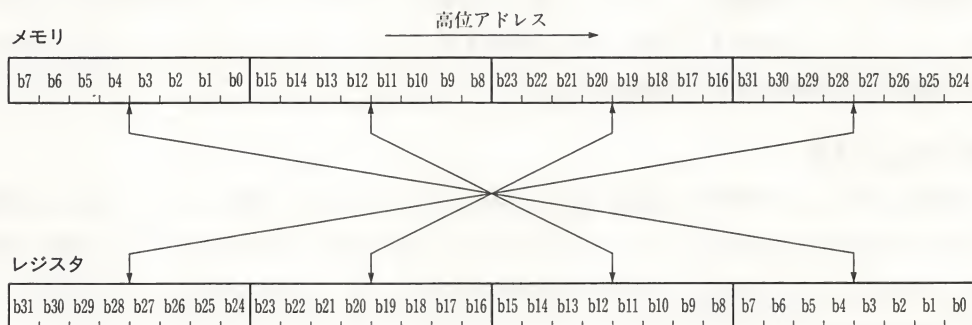
b15 b14 b13 b12 | b11 b10 b9 b8　　b7 b6 b5 b4 | b3 b2 b1 b0　16ビットデータ

高位アドレス →

80386の場合

b7 b6 b5 b4 | b3 b2 b1 b0　8ビットデータ

b7 b6 b5 b4 b3 b2 b1 b0　　b15 b14 b13 b12 | b11 b10 b9 b8　16ビットデータ

高位アドレス →

▼図I-2-11　32ビットデータを転送するときのレジスタとメモリのビット対応

2.5　仮想記憶と物理アドレスの生成

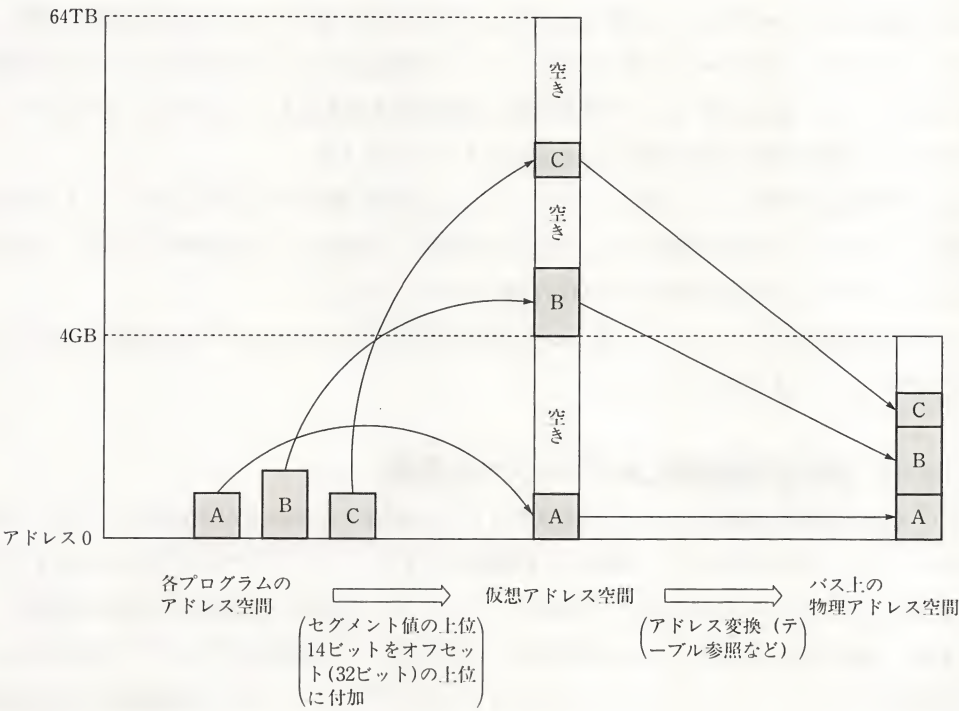
2.5.1　80386の仮想記憶の概念

一般に，仮想記憶といえば，アプリケーションプログラムに対して，バスのアドレス空間を拡張する手法と考えがちです．たしかに，8ビットCPUの時代は，その目的で仮想記憶が利用されることが多かったようです．しかし，80386では少し違って，マルチタスクや記憶保護と大きくかかわっています．

複数のプログラムが並行して実行されるとき，あるプログラムがメモリにロードされるアドレスは，そのときのシステムの状況，いわば偶然に左右されます．すなわち，同時に実行される他のプログラムのサイズによって，その位置は変化します．言い換えると，個々のプログラムは，ロードアドレスに依存せずに実行できるようでなければなりません．

また，個々のプログラムは，お互いに無関係に動作するのが原則ですから，終了する時点もそれぞれ異なります．終了したプログラムが使用していたアドレス空間は解放されて，別のプログラムに割り当てられなければなりません．そのためには，メモリの管理が容易でなければならないのです．

80386の仮想記憶の考え方を，図I-2-12に示します．

▼図I-2-12　80386の仮想記憶の概念

通常，80386では，個々のプログラムは，アドレス空間の低い位置を設定して作成します．そして，実行段階ではセグメントレジスタ(その中のアドレス情報14ビット)をプログラムの番号のようにして使用し，オフセットの32ビットと併せて46ビットの仮想アドレス空間(64TB)を作ります．この処理は，「セグメンテーション」と呼ばれます．こうすると，全部のプログラムが，オフセット0番地から始まっていても，仮想アドレス空間では大きく離れていますから，衝突する心配がなくなります．したがって，論理アドレス空間は広くても，現実にはすき間だらけで運用されます．

この方法は，一見，面倒なようですが，プログラムを作成する際に，実際にロードするアドレスを意識しなくてもすみます．

しかし，このようなアドレス空間は，あくまでも仮のものです．実際にCPUが直接アクセスできるバス上の空間が32ビットのアドレス線によるものである以上，4GBの枠の中で「タライ回し」せざるをえません．

そこで，仮想アドレス値を基に，テーブルを参照し，バス上の物理空間に変換します．このとき，テーブルを参照する前に，46ビットの仮想アドレス値を32ビットに圧縮します．この段階での32ビットアドレス値を，「中間リニアアドレス」といいます．

中間リニアアドレスは，上位20ビットと下位12ビットに分割されます．上位のビットは，テーブル参照に使用され，変換値に置き換えられてから，アドレスバスに出力されます．いわば，アドレスの「組み換え」が行われるわけです．下位の値は，そのままアドレスバスに出力されます．

この操作は，「ページング」と呼ばれます．下位12ビットでアクセスできる空間は4KBですから，上位20ビットをページと考えれば，ページと物理アドレスとの対応をテーブルで制御することによって，任意のブロック(4KB単位)位置に置き換えられるわけです．すなわち，4GBのアドレス空間は4KB単位に割り当てできるようになります．

以上の動作を利用すると，終了したプログラムの位置に別のプログラムをロードして実行する際に，空きエリアが飛び飛びでも，あたかも連続した空間のように処理できます．なぜならば，リニアアドレスの上位20ビットは組み換えできるからです．

メモリの内容をディスクにセーブ／ロードすれば，同じメモリブロックを異なったアドレスに再配置することも可能になります．

2.5.2　仮想記憶管理と実アドレスへの変換

ここでは，80386が論理アドレスを物理アドレスに変換する過程を説明します．なお，これから述べるアドレス生成時には，同時に，不適切なアドレスへのアクセスが行われないようにするための，チェックが行われます．したがって，アドレス生成と保護機能は密接に連携していますが，本節では，まずアドレス生成に関してだけ述べ，保護機能に関しては次節で扱います．

80386は，32ビットのリニアアドレスによって，4GB(ギガバイト)までの物理アドレスを直接

アクセスできます．さらに，仮想記憶管理を使用すれば，64TB(テラバイト)までの論理アドレスに対応することができます．ただし，論理アドレスはあくまでもCPU内部でのアドレスです．バスによってアクセスされる物理アドレス(32ビット)の総和が実際に有効な領域のサイズとなります．

アドレス生成の流れの概略を，図I-2-13に示します．

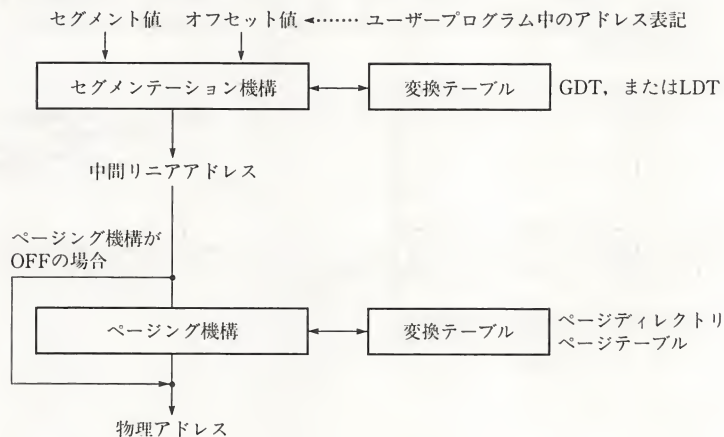
論理アドレスは，CPUの「セグメンテーション機構」によって，「中間リニアアドレス」に変換されます．変換に際しては，変換テーブル(GDT，またはLDT)が使用されます．中間リニアアドレスは，32ビットの幅です．

中間リニアアドレスは，さらに「ページング機構」によって最終的な「物理アドレス」に変換されます．この変換に際しては，2種類の変換テーブル(ページディレクトリ，ページテーブル)が使用されます．

TownsOSは，セグメンテーション機構とページング機構の両方を利用しています．

なお，ページング機構は，ソフトウェアによってバイパス(OFF)できるので，セグメンテーション機構だけを利用することも可能です．この場合には，中間リニアアドレスがそのまま物理アドレスになります．

▼図I-2-13　80386のアドレス生成の流れ

セグメント値　オフセット値 ◀…… ユーザープログラム中のアドレス表記
セグメンテーション機構 ⟷ 変換テーブル　GDT，またはLDT
中間リニアアドレス
ページング機構がOFFの場合
ページング機構 ⟷ 変換テーブル　ページディレクトリ　ページテーブル
物理アドレス

●セグメンテーション機構

プログラム中でのアドレスは，「セグメント」と「オフセット」によって表記されています．これらは，まずセグメンテーション機構によって処理されます．この機構の動作は，CPUの動作モードによって異なりますが，ここではFMTOWNSの動作しているプロテクトモードに関して述べます．

プロテクトモードでは，セグメントは「ディスクリプタテーブル」と呼ばれる表によって管

理されています．ディスクリプタテーブルには，個々のセグメントを管理する「ディスクリプタ」が格納されています．セグメントレジスタの内容は，テーブル中の何番目のディスクリプタを使用するかを指し示すインデックスであり，「セレクタ」と呼ばれます．

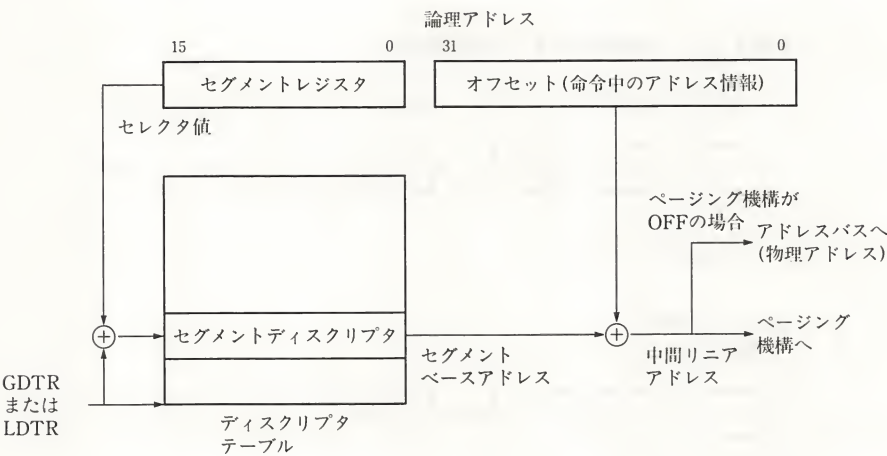
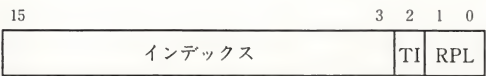
この過程の概略を，図I-2-14に示します．

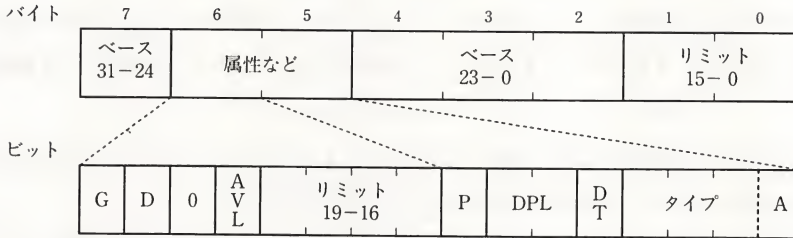
セレクタの構成を図I-2-15に示します．

図I-2-16に，ディスクリプタの構成を示します．このディスクリプタを集めた表が，ディスクリプタテーブルです．ディスクリプタテーブルには，グローバルディスクリプタテーブル(GDT)と，ローカルディスクリプタテーブル(LDT)の2種類がありますが，構成は同一です．

ベースアドレスとリミットが，分散して記録されているのは，80286との互換性を保持するためです．

GDTは，システム上に1つだけ存在するディスクリプタテーブルです．GDTによって管理されるセグメントには，OSの各種サービスルーチンや，割り込みハンドラ用のテーブルなどを置きます．

▼図I-2-14　プロテクトモードのセグメント処理

▼図I-2-15　セレクタ

インデックス(13ビット)	：ディスクリプタテーブル中のディスクリプタを指し示す番号．
TI(Table Indicator)	：GDTかLDTのどちらをアクセスするかを示す． 0＝GDT 1＝LDT
RPL(Requestor's Privilege Level)	：要求特権レベル． 保護機能によって使用される．

▼図Ⅰ-2-16　ディスクリプタ

ベースアドレス(32ビット)　：そのセグメントの先頭アドレス．中間リニアアドレスで設定する．

リミット(20ビット)　：そのセグメントのオフセットの最大値(上限値または下限値)を保持する．保護に用いる．値の単位は１バイトまたは4KBで，後述のＧビットで指定する．

G(Granularity)　：リミット設定単位．リミットの単位を１バイト／4KBのいずれにするかを指定する．
０＝１バイト単位
１＝4KB単位

D(Default)　：80286のプロテクトモードとの互換性の有無を指定する．
０＝互換性あり．アドレスとオペランドのデフォルトのサイズが，80286のプロテクトモードと同じように行われる．
１＝互換性なし．デフォルトは32ビットのアドレスと32ビットと８ビットのオペランドになる．

AVL(Available to Software)　：ユーザープログラムに解放されている．このビットはCPUに，何ら影響を与えない．

P(Present)　：そのセグメントが，現在，物理メモリ上に存在しているか否かを表す．CPUは，P＝0のセグメントがアクセスされると，例外を発生する．OSの例外ハンドラは，補助記憶装置(ハードディスク)から，そのセグメントをロードするなどし，その後このビットを１にする．
０＝無効．存在していない．
１＝有効．存在している．

DPL(Descripter Privilege Level)：セグメントの特権レベル．保護機能によって使用される．

DT(Descripter Type)　：セグメントの種類の指定
０＝システムセグメント，またはゲート
１＝メモリセグメント

タイプ　：アクセスタイプなどの指定．
０＝リードのみ可能なデータセグメント
１＝リード／ライト可能なデータセグメント
２＝リードのみ可能な下方伸長型データセグメント
３＝リード／ライト可能な下方伸長型データセグメント
４＝実行のみ可能なコードセグメント
５＝リード／実行可能なコードセグメント
６＝実行のみ可能な適応型コードセグメント*
７＝リード／実行可能なコードセグメント*
　*では，特権レベルのチェックが省略される．

A(Access)　：アクセスビット．そのセグメントがアクセスされると，このビットがセットされる．OSで定期的にこのビットをテストしクリアすることで，そのセグメントの使用頻度を調べることができ，使用頻度の低いセグメントをディスクに待避するように，OSをデザインできる．

CPUのシステムレジスタの1つであるGDTRは，このGDTのアドレスを保持するレジスタです．GDTRの値は，システムの起動時にOSによって設定されます．

LDTは，タスクごとに使用するディスクリプタテーブルで，CPUがタスクを切り換えるとLDTも切り換えられます．LDTは，それ自身が1つのセグメントとして，GDTに登録されています．

各タスクは，そのタスク専用のLDTを通してセグメントをアクセスします．したがって，そこに登録されていないセグメントはアクセスすることができません．

CPUのシステムレジスタの1つであるLDTRは，このLDTのGDT上での位置を示しています．

セグメントの処理では，ディスクリプタテーブルの参照が必要ですが，頻繁に参照が行われるとメモリのアクセスが多くなり，実行速度が遅くなります．そこで，80386では，セグメントレジスタごとに，4バイトのキャッシュを用意して，ディスクリプタ(4バイト)をCPU内に保持するようにしています．このため，キャッシュによってディスクリプタを切り換えずに済む場合には，メモリアクセスが不要となります．

●ページング機構

ページング機構では，論理アドレス空間は，一定の大きさの「ページ」に分割されて管理されます．1ページの大きさを一定にすることによって，論理アドレスをできるだけ無駄なくギッシリと物理アドレス空間に割り当てることができます．

80386の扱う1ページの大きさは4KBで，64TBの論理アドレス空間は，16M(メガ)個のページに分割されます．16M個のページは，ページテーブルディレクトリとページテーブルという，2種類のテーブルによって管理されます．

セグメンテーション機構で生成された中間リニアアドレスは，次のようなページング機構によって，最終的な物理アドレスに変換されます．その過程を，図I-2-17に模式化して示します．

ページング処理では，前述の中間リニアアドレス(32ビット)が10＋10＋12ビットの形で3分割され，次の3段階の手続きが行われます．

1．第1の手続きでは，リニアアドレス上位10ビットをインデックスとして用い，ページテーブルディレクトリ(図I-2-18)を参照して，次に引くページテーブルのベースアドレスを取得します．

ページテーブルディレクトリの物理アドレスは，CR3の上位20ビットに格納されています．20ビットで済むのは，1ページのサイズが，4KBに固定されているためです．ページテーブルディレクトリは，すべてのページを管理する根本となるテーブルで，必ず物理メモリ上に存在していなければなりません．

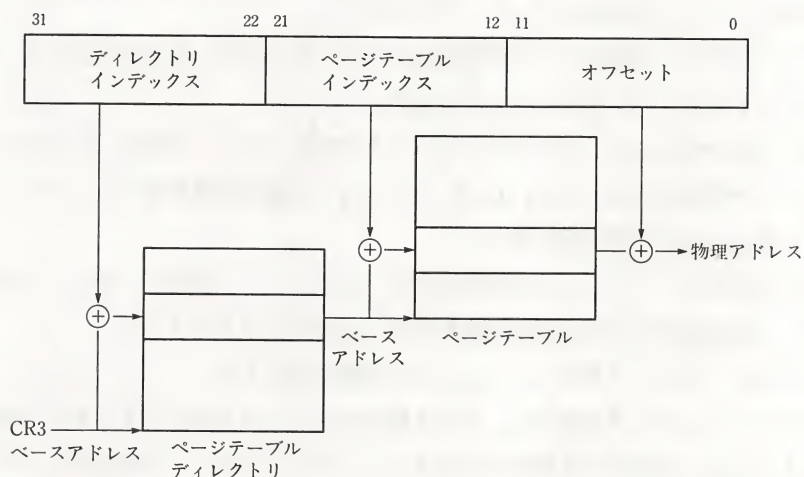
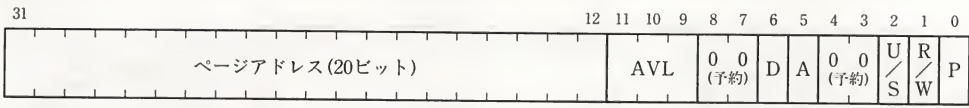
ページテーブルディレクトリ中の各項目は，32ビットの長さです．

2．第2の手続きでは，このページテーブルディレクトリと同じフォーマットのページテーブルを参照し，中間リニアアドレスの中ほどの10ビットにより，位置決めしてページアドレスを読み出します．

3．最後の手続きで，目的のページのベースアドレスに，中間リニアアドレスの下位12ビットをオフセットとして加算すると，最終的な物理アドレスが得られます．

▼図Ⅰ-2-17　ページング処理の概念

▼図Ⅰ-2-18　ページテーブルディレクトリ

31–12	11–9	8–7	6	5	4–3	2	1	0
ページアドレス(20ビット)	AVL	0 0 (予約)	D	A	0 0 (予約)	U／S	R／W	P

P (Present)：該当するページが，物理メモリ上に存在しているか否かを示す．
0＝存在していない
1＝存在している
0の場合には，CPUはページフォールトエラーを通知する例外を発生する．この場合には，1～31ビットは，ソフトウェア(OS)が利用できるAVL(Software Available)と見なされる．

ページアドレス(20ビット)：次に参照するページテーブルが，何ページ目にあるかを示す．ページの長さは，4KBに決まっているので，20ビットあればよい．

R／W (Read／Write)：ページ単位の保護に関するビット．
0＝リードのみ可能なページ
1＝リード／ライトともに可能なページ

U／S (User／Superuser)：そのページを使用しているユーザーのレベルを表す．ページレベルでの保護．
0＝スーパーユーザー
1＝ユーザー
特権レベルが3のコードは，U／S＝0のページをアクセスできない．
U／S＝1のページへは，R／Wの設定に従ったアクセスが可能．
特権レベル2～0のコードは，アクセスの制限を受けない．

A (Accessed)：そのページへのアクセスがあった場合に，CPUがセットする．

D (Dirty)：そのページへ書き込みが行われると，そのページに関するページテーブル中のエントリのこのビットが，CPUによってセットされる(クリアすることはない)．

AVL (Software Available)：このビットは，CPUに何ら影響を与えない．ソフトウェアで利用することができる．

仮想記憶で扱える64TBを，4KBのページに分割すると，2の20乗ページになります．1つのページの管理に4バイトのデータが必要ですから，ページの管理だけで最大4MBのデータが必要になります．これほどの管理データを物理メモリ上に保持するのは，現実的ではありません．

そのために，80386では，ページの管理をページディレクトリとページテーブルという，2つのテーブルに分けて行っています．第1のテーブル(ページディレクトリ)は物理メモリ上に存在していなければなりませんが，第2のテーブル(ページテーブル)はそれ自身がページなので，必ずしも物理メモリ上に存在していなくてすみます．

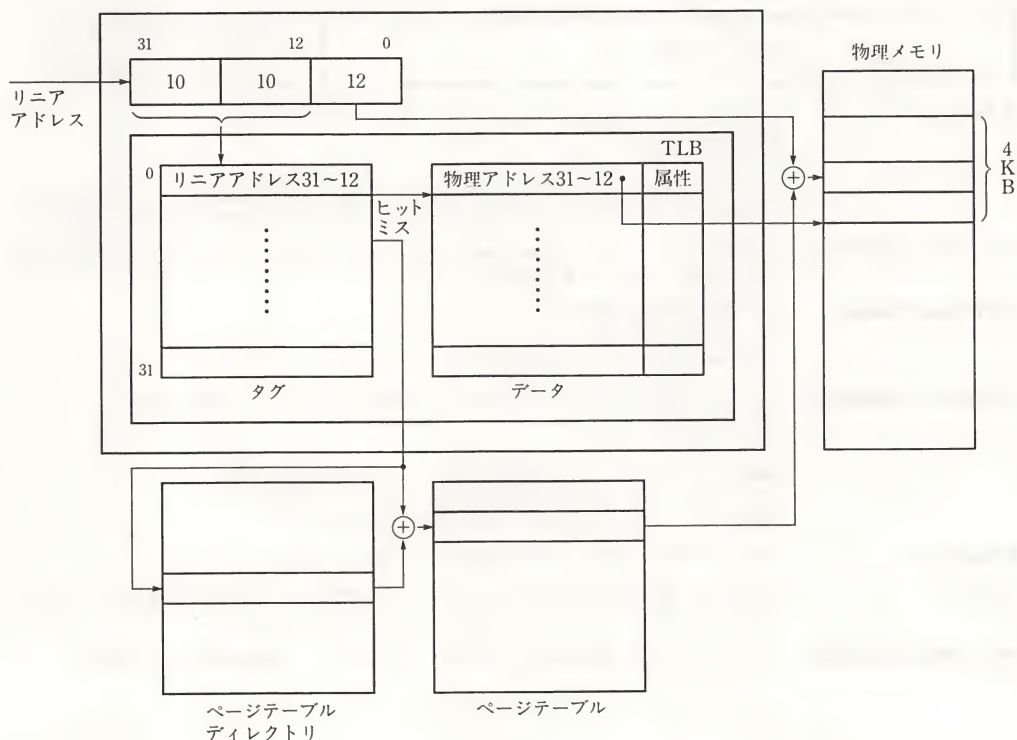
ページテーブルと，目的のページが物理メモリ上に存在しているか否かは，ページディレクトリのPビットをテストすることで分かります．

P＝0ならば，そのページ(ページテーブルも)は物理メモリ上に存在しないので，CPUはページフォールト例外を発生します．OSは，そのページを補助記憶装置(ハードディスク)から物理メモリに移すなどの処理を行います．

このことを利用して，ディスクの領域を見かけ上のアドレス空間の一部として使うことができますが，入出力を伴なうため処理速度が低下するのはやむを得ません．

P＝1ならば，Aビットをセットして，次の処理に進みます．

このように，ページング処理では，必ず2種類のテーブルを参照しますが，両者とも物理メモリ上にあっても，参照のため段階的に読み出しを行うのでスピードが低下します．

▼図I-2-19　中間リニアアドレスから物理アドレスへの変換

そこで，80386は高速化を計るために，ページング処理用のバッファを内部に備えています．これは，トランスレーションルックアサイドバッファ(TLB)と呼ばれ，最近アクセスした32個までのページテーブル項目を格納しています．

TLBも含めたページングの概念を，図I-2-19に示します．

2.6　保護機能

80386の記憶保護は，4段階の特権レベルからなる「リング型保護」と，タスク間相互の影響を排除する「タスク間保護」の両面から行われています．前者は上下方向の保護，後者は横方向の保護といってもよいでしょう．

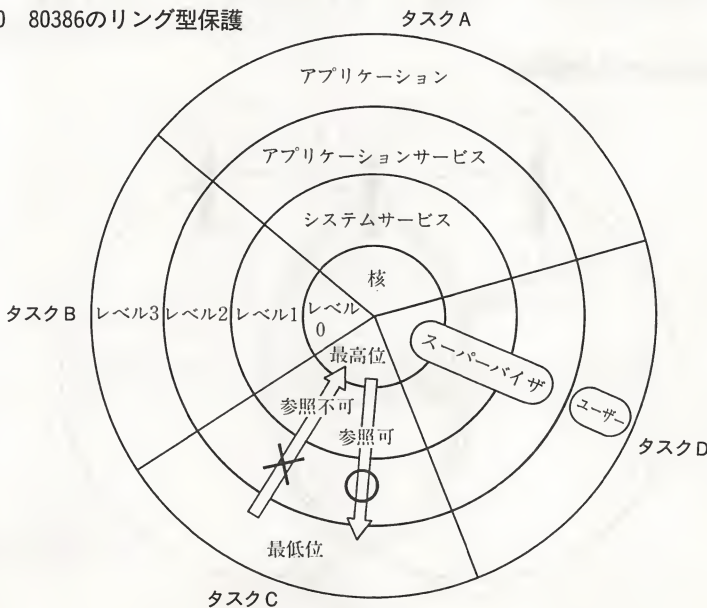
2.6.1　リング型保護

リング型保護は，タスクに「特権レベル」という順位を設け，この特権レベルによってメモリアクセスや実行できる命令を制限する方式の保護です．

リング型保護の概念を，図I-2-20に示します．

80386では，特権レベルは4レベルあります．特権レベル0が最高の優先度を持ち，特権レベル3が最も優先度の低いレベルです．上位の特権レベルから下位のメモリは参照できますが，その反対はできません．このチェックは,CPUが論理アドレスから物理アドレスを生成する際に，同時に行われます．

リング型保護の形態としては，メモリアクセスに関する保護だけでなく，実行を制限する「特権命令」があります．

▼図I-2-20　80386のリング型保護

特権命令とは，特権レベル0だけに許される命令です．CPUの制御などの重要な命令は，OSにしか実行を許さないようにし，下位のレベルのプログラムで使用させないようにするもので，これも他のタスクに影響を及ぼす可能性のある動作を排除するものです．

リング型保護は，マルチタスクOSでは重要な機能です．タスクに割り当てるメモリの管理テーブルのようなシステムに重要なデータへは，特権レベルの高いOSでなければアクセスできないようにし，普通のプログラム(ユーザープログラムやアプリケーションプログラム)が暴走しても，システムに重大な被害が及ばないようにデザインされます．

なお，CPUは4レベルの特権レベルを用意していますが，OSのデザインの仕方によって，すべてを利用する必要はありません．OSを特権レベル0とし，普通のプログラムを特権レベル3として，2レベルだけを利用するOSもあります．シングルタスクのOSであれば，特権レベル0だけで動作するデザインも考えられます．

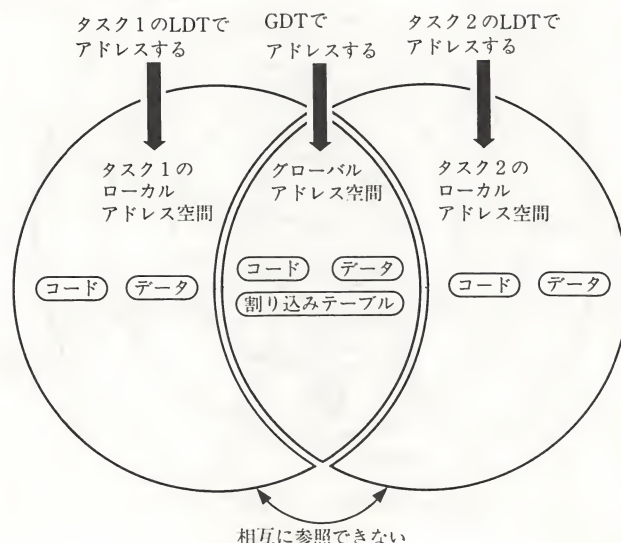
2.6.2 タスク間保護

タスク間保護は，特権レベルの同じタスク同士でも，あるタスクの影響が他へ及ばないようにするための保護です．いわば，リング保護が上下方向なのに対し，これは横方向への保護策です．

タスク間保護は，タスクごとに固有のLDTを持つことによって行います．この概念を図Ⅰ-2-21に示します．セグメントを管理するLDTのディスクリプタには，セグメントのサイズが格納されており，このセグメントリミット(サイズ)を越える参照を阻止することによって，他のタスクのセグメント領域を侵害しないようにするものです．

このチェックも，CPUのアドレス生成時に同時に行われます．

これらの記憶保護によって，マルチタスク動作時に，あるタスクまたはセグメントのトラブルが他のタスクや上位のセグメントを巻添えにすることを防ぐことができます．

▼図Ⅰ-2-21　80386のタスク間保護

2.6.3 タスク内での保護

80386は，同一タスク内でセグメントとページに対するアクセスを制限する，「アクセスライト」という保護機能も提供しています．アクセスライトは，次のように設定できます．これに違反すると，プロテクションエラーとなり，一般にOSに制御がもどされます．

●セグメント

コードセグメント：リード可能／不可能

データセグメント：リード可能／不可能

●ページ(特権レベル3のみ)

リードオンリー／リード・ライト可能

特権レベル0～2でのページのアクセスライトは，常にリード・ライト可能となっています．

2.7 特権レベルの切り換えとゲート

リング型保護やタスクの切り換えに際しては，例えば，特権レベルの低いアプリケーションプログラムが，特権レベルの高いOSのサービスを受ける(OSの機能を利用する)ようなときに，特権レベルを切り換える手段が必要になります．しかし，普通のジャンプ命令やコール命令では，特権レベルの異なるプログラムへ制御を移すことはできません．

▼表 I-2-4　特殊ディスクリプター覧

TYPE	種　別
0	未使用
1	待ち状態の286TSS
2	ローカルディスクリプタテーブル
3	実行中の286TSS
4	286コールゲート
5	タスクゲート
6	286割り込みゲート
7	286トラップゲート
8	未使用
9	待ち状態の386TSS
10	未使用
11	実行中の386TSS
12	386コールゲート
13	未使用
14	386割り込みゲート
15	386トラップゲート

80386では，記憶保護のリングやタスクを乗り越えることのできる機能として，「ゲート」が用意されています．ゲートは，特殊ディスクリプタの1つです．80386のすべての特殊ディスクリプタを，表I-2-4に示します．

ゲートには，次の4種類があります．

コールゲート……高い特権レベルに移行
タスクゲート……他のタスクに移行
割り込みゲート…割り込みハンドラに移行
トラップゲート…例外ハンドラに移行

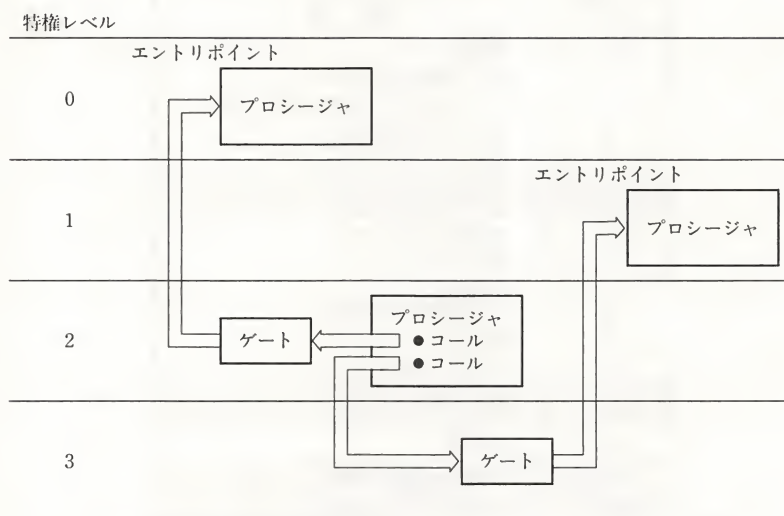
これらのゲートの中で，特権レベル間の移行には，コールゲートが使われます．

ひとつのタスクを，異なった特権レベルのプログラム群によって，構成することもできます．

ゲートの使用方法は，コール命令のセレクタでコールゲートを参照し，それによって目的のプロシージャ(プログラム内ブロック)を呼び出します．言い換えると，コールゲートの内容から飛び先のアドレス(プロシージャのエントリポイント)を組み立てます．

この関係を特権レベルの移行という観点からとらえると，図I-2-22のようになり，飛び先アドレスの生成という観点から説明すると，図I-2-23のようになります．

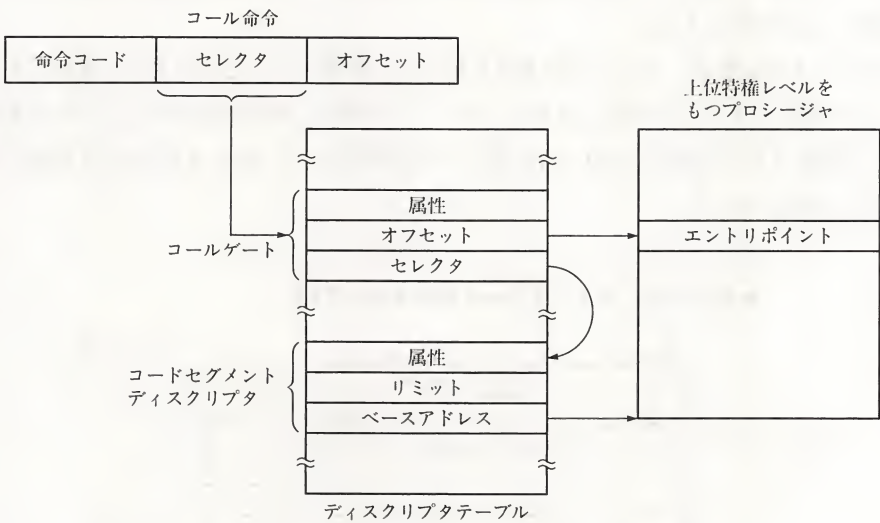
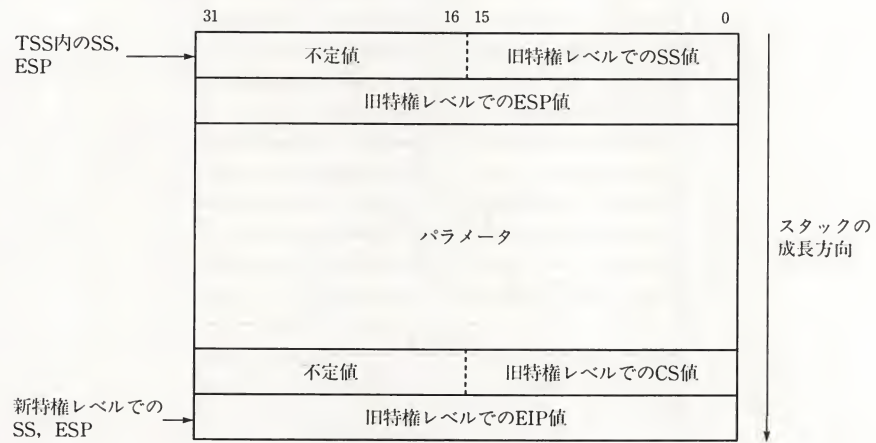
80386は，特権レベルごとにスタックを用意しています．特権レベルの切り換え時には，このスタックを介して，特権レベル間でデータが受け渡しされます．移行の際は，旧特権レベルで使用していたスタック関連のレジスタ(SS，ESP)の値を，新特権レベル側のスタックへとセーブします．新スタックには，さらに，旧特権レベルのスタックから，コールゲートの属性で指

▼図I-2-22 コールゲートを使用した上位特権レベルへの移行

定されたワード数のパラメータもコピーされ，最後に旧特権レベルのEIPがセーブされます．スタックの構成を，図I-2-24に示します．

このようなスタックによって，元の特権レベルへ復帰するときにタスクを復元できます．

復帰時には，RET n 命令(nは，スタック上のパラメータのバイト数)を使用すると，新特権レベルに残っているパラメータを破棄してリターンします．このときEIPは元の値に復元され，スタックポインタ(SS，ESP)がもどされ，旧特権レベルにコピーされたパラメータも破棄されます．EIPが復元されることによって，コール命令の次の命令から実行が再開されます．

▼図I-2-23　コールゲートによる飛び先アドレスの生成

コール命令
命令コード
セレクタ
オフセット
上位特権レベルを
もつプロシージャ
属性
オフセット
セレクタ
コールゲート
エントリポイント
属性
リミット
ベースアドレス
コードセグメント
ディスクリプタ
ディスクリプタテーブル

▼図I-2-24　より高い特権レベルに移行した直後のスタック

2.8 タスク間の移行

マルチタスク環境でタスクを切り換える際には，各タスクのレジスタの値などの情報を保存しておかなければ，そのタスクを再開できなくなってしまいます．

80386の各タスクは，タスクごとに，実行情報を格納する「タスクステートセグメント(TSS)」を持っています．TSSには，タスクの属性，レジスタ群の値，バックリンクが保存されます．TSSの構成を，図I-2-25に示します．

タスクへの切り換えには，「タスクゲート」を使用します．タスクゲートによってTSSディスクリプタを参照し，新しいTSSを選定することによってタスクが切り換えられます．この概念を，図I-2-26に示します．

タスクゲートの起動は，ジャンプ命令またはコール命令によって行います．そのようすを，図I-2-27に示します．このとき，旧タスクのレジスタ群は，旧TSS内にセーブされます．また，新しく選定されたTSSからは，各レジスタの内容がロードされ，EIPが示す位置から新タスクの実行が始まります．

▼図I-2-25　タスクステートセグメント(TSS)

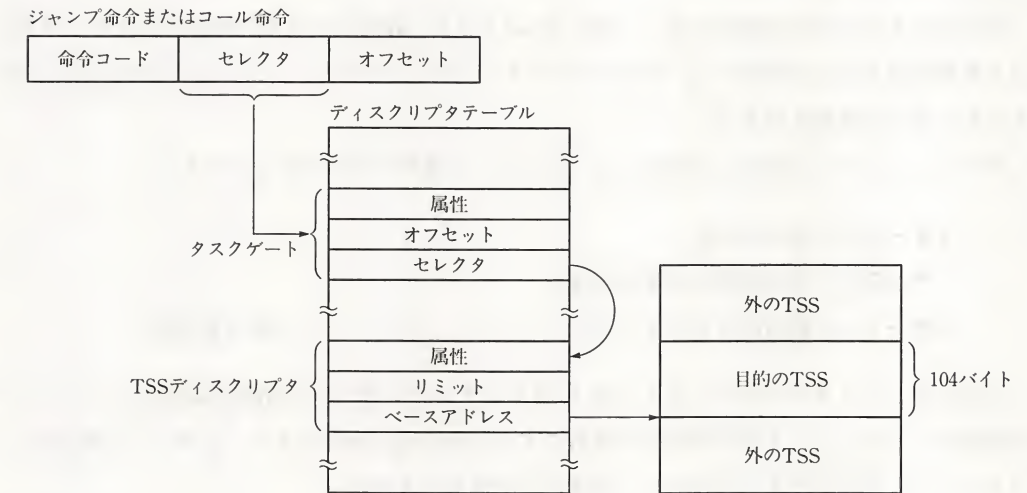
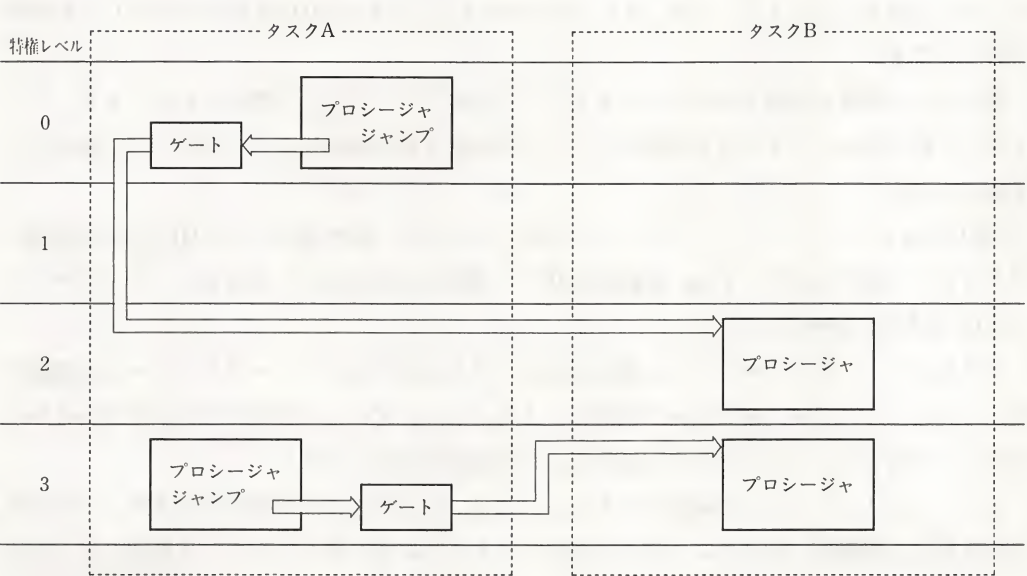
31　　　　16	15　　　　0
タスクの属性	
0	LDTR
0	GS
0	FS
0	DS
0	SS
0	CS
0	ES
EDI	
ESI	
EBP	
ESP	
EBX	
EDX	
ECX	
EAX	
EFLAGS	
EIP	
CR3	
0	SS2
ESP2	
0	SS1
ESP1	
0	SS0
ESP0	
0	バックリンク

アドレス上位（↑）

合計104バイト

タスクの切り換えには，もっと高速な方法もあります．それは，TSSディスクリプタを直接指すセレクタを使用して，ジャンプまたはコール命令を実行する方法です．ただし，この方法では，切り換え先のタスクが限定されます．

いずれの方法でも，タスクの切り換えはハードウェアによって行われるため，時間の損失は最小限ですみます．特に，後者のタスクゲートを使用しない方法では，タスクの切り換えに要する時間は，17μ秒ですみます．

▼図I-2-26 タスクゲートによる新TSSの参照

▼図I-2-27 タスクゲートの起動とタスクの移行

2.9　割り込みと例外

現在実行中のタスクを中断して，別のタスクを起動するきっかけには，「割り込み」と「例外」があります．両者の違いは，原因が周辺デバイスで発生するか，CPU内部で発生するかという点です．割り込みは，周辺デバイスがCPUに対して処理を要求するときに発生します．例外は，命令の実行中に何らかの異常や特定の状態が検出されると発生します．これらが発生すると，現在実行中のタスクが自動的に中断され，発生要因が特定された後，異常などを排除する処理が行われます．

割り込みまたは例外が発生すると，表I-2-5のように，最初にベクタが特定されます．INTRによる外部割り込みの場合には，8259(プログラマブルインタラプトコントローラ)によって与えられた番号が採用されます．

表I-2-5において，例外の「種別」は，次のような基準で分けられています．

フォールト…再実行可能
トラップ……次の命令から実行可能
アボート……致命的なOSまたはハードウェア上のエラーで，復元不可能

フォールトとトラップでは，CSとEIPの値をスタックにセーブする際の命令を，フォールトは現命令のものにし，トラップは次の命令にするという違いがあります．アボートの場合は，リセットによる再立ち上げ以外には，回復の方法はありません．

ベクタ番号が特定されると，80386は，この値をインタラプトディスクリプタテーブル(IDT)のインデックスとして使用し，このテーブルに登録されているゲートの1つを起動します．テーブルに登録されているゲートを，表I-2-6に示します．IDTは，IDTRの示すアドレスに配置されています．

割り込みや例外を処理するプログラムを，ここでは「ハンドラ」と呼ぶことにします．ゲートから，割り込みハンドラまたは例外ハンドラが起動される過程は，コールゲートの場合によく似ています．

「割り込みゲート」と「トラップゲート」は似ていますが，割り込みフラグ(IF)の処理に違いがあります．割り込みゲートは，起動時にIF＝0(割り込み禁止)にしますが，トラップゲートは，IFを変更しません．

これらのゲートがコールゲートと異なる点は，スタック中のパラメータをコピーする機能を持っていないことです．割り込みや例外のように，いつ，どのような状態で発生するか分からないものに対して，パラメータを用意することは無意味だからです．

また，これらのゲートから起動されたハンドラは，同一タスク内で実行されます．すなわち，メモリなどの資源は，割り込みや例外が発生したタスクに割り当てられていた範囲のものが使

用できます．

一方，タスクゲートでの割り込みフラグの処理は，TSS中のIFフラグの状態によります．また，タスクゲートで起動されたハンドラは別タスクとなります．元のタスクのメモリなどを参照することができなくなるので注意が必要です．

▼表Ⅰ-2-5　割り込みと例外対応のベクタ

ベクタ	種　別	例外状態
0	フォールト	除算エラー
1	トラップ/フォールト*1	デバッグ例外
2	割り込み	NMI割り込み
3	トラップ	ソフトウェアブレークポイント
4	フォールト	オーバーフロー
5	フォールト	配列境界チェック
6	フォールト	無効オペコード
7	フォールト	コプロセッサ不在
8	アボート	システムエラー
10	フォールト	無効TSS
11	フォールト	セグメント不在
12	フォールト	スタックオーバーフロー/アンダーフロー
13	フォールト	一般プロテクション例外
14	フォールト	ページ不在
16	フォールト	コプロセッサエラー
0～255*2	割り込み	INTRによる割り込み

＊1「2.10 デバック機能」を参照のこと．
＊2 INTRのベクタは通常8259Aから受け取る．8259Aが出力するベクタはプログラマブルになっており，ソフトウェアで設定する．

▼表Ⅰ-2-6　割り込み，または例外時の処理で使用されるゲート

型	ハンドラの型	ゲート通過後の割り込みの許可/禁止
割り込みゲート	プロシージャ	禁止(IF＝0)
トラップゲート	プロシージャ	変化なし，IFの値は割り込み，または例外発生以前の値を保持
タスクゲート	タ　ス　ク	TSS中のIFフラグによる

2.10 デバッグ機能

80386は，デバッグのために，シングルステップ実行とブレークポイント機能を，ハードウェア化して内蔵しています．

デバッグ機能に関係するデバッグレジスタには，DR0〜DR7があります．デバッグレジスタの内容を，図Ⅰ-2-28に示します．これらのレジスタをアクセスするためには，特権レベルが0となっていることが必要です．

▼図Ⅰ-2-28 80386のデバッグレジスタ

31　　　　15　　　　0	
ブレークポイント0(中間リニアアドレス)	DR0
ブレークポイント1(中間リニアアドレス)	DR1
ブレークポイント2(中間リニアアドレス)	DR2
ブレークポイント3(中間リニアアドレス)	DR3
インテルによって予約済	DR4
インテルによって予約済	DR5
デバッグステータスレジスタ	DR6
デバッグコントロールレジスタ	DR7

デバッグのための例外ハンドラは，ベクタ番号1の例外によって起動され，トラップゲートを使用します．タスクゲートでは，タスクが切り換えられてしまうために，ターゲットプログラムの状況が参照できなくなります．また，割り込みゲートでは，割り込みのマスクのために，周辺装置を駆動するのが困難になるなどの支障が生じてしまいます．

シングルステップ実行によるデバッグでは，プログラムを完全にモニタすることができます．シングルステップ実行を行うためには，EFLAGS(フラグレジスタ)のTF(トレースフラグ)を1にします．これによって，CPUは，命令を1つ実行するたびに例外を発生するようになります．ただし，例外ハンドラは，リターンするまで連続的に実行されます．もし，例外ハンドラまでもシングルステップ実行されると，収拾がつかなくなるだけでなく，ステップ実行の目的からも外れてしまいます．

ブレークポイントは，4個まで，ブレーク条件とともに設定できます．

ブレークポイントのアドレスは，DR0〜DR3に，32ビットのリニアアドレスで設定します．ブレーク条件は，ブレークポイントごとに，アクセスのタイプとモニタするデータの長さを，DR7(デバッグコントロールレジスタ)に設定します．

アクセスのタイプとしては，次の3種類の1つを選択できます．

・命令実行時のみに例外を発生する．
・データ書き込み時のみに例外を発生する．
・データ読み込み時，またはデータ書き込み時に例外を発生する．

モニタするデータの長さは，ブレーク範囲とか，ブレークポイントのサイズとも呼ばれるものです．設定したブレークポイントのアドレスから，１，２，４バイト(いずれかを選択)の範囲のメモリがアクセスされると，例外が発生します．

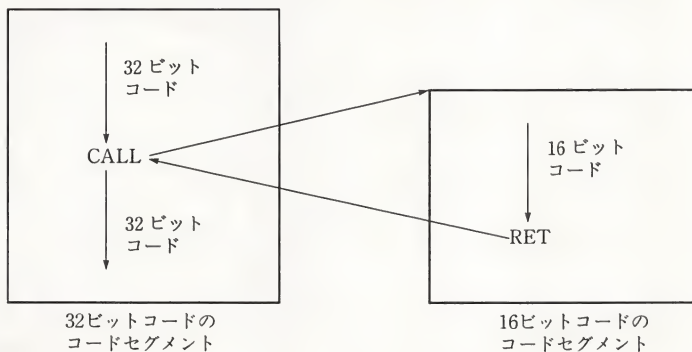
例外が発生すると，DR6(デバッグステータスレジスタ)のブレークポイントに対応したフラグが立ちます．例外ハンドラは，このレジスタを参照することによって，どのブレークポイントで例外が発生したか特定することができます．DR6には，シングルステップ実行時の例外発生，タスク切り換え時の例外発生を通知するフラグもあります．

また，マルチタスク環境では，タスクごとにブレークポイントを設定できなければなりません．TSSのタスク属性のデバッグフラグを１にしておくと，タスク切り換え時にも例外が発生するようになります．この例外によって，タスク切り換え時に，デバッグレジスタの内容を書き換えることができます．

2.11　プロテクトモードでの16ビットコードの実行

80386では，プロテクトモードでも，16ビットコードを実行できる機能があります．具体的には，コードセグメントディスクリプタの属性フィールドのDビットを０にすると，16ビットコードで実行することができます．D＝１のとき，本来の32ビットコードで働くようになっているわけです．

したがって，セグメント単位で，16ビット／32ビットコードの選択を行うことができ，混在させることも可能です．このようすを，図Ｉ-2-29に示します．

▼図Ｉ-2-29　16ビットコードと32ビットコードセグメントの混在

ただし，80386のプロテクトモードでは，セグメントレジスタの動作が8086とは異なるために，そのままでは8086のオブジェクトレベルでの互換性を得ることはできません．この問題を解決するために，80386では「仮想8086モード」がサポートされています．

プロテクトモードから仮想8086モードに切り換えるには，EFLAGS(フラグレジスタ)中のVMを1にします．VMの操作には，特権レベルが0であることが必要で，スタックに待避されたEFLAGSの部分，またはTSS中のEFLAGS部分を重ね書きすることによって切り換えることができます．なお，VMを1にすると，16ビットコードで実行されます．

いったん仮想8086モードになると，フラグレジスタ(EFLAGS)は，下位8ビットのEFLAGSの部分しか参照できなくなります．VMはEFLAGSの上位8ビット中にありますから，仮想8086モード上のタスク自身がVMを書き換えて，プロテクトモードにもどることはできなくなります．

しかし，仮想8086モードはプロテクトモードの一部として動作しますから，例外や外部割り込みの発生によって，自動的にプロテクトモードにもどることができます．

仮想8086モードは，常に特権レベル3に置かれます．したがって，オブジェクトレベルで8086と互換性があるといっても，特権命令であるI/O命令を使用すると例外が発生します．このときのハンドラを特権レベル0に置き，I/O処理部分をプロテクトモード側でシミュレートするといった程度の補足は必要です．

第 3 章

CPU近傍のデバイス

FMTOWNSのCPUの近傍には,CPUと周辺装置とのインタフェースを補助するデバイスなどがあります．その中でも重要な働きをしている割り込み関係，DMA関係，タイマとクロック関係のデバイスについて解説します．また，章の終わりでは，CPU近傍に配置されている各種のレジスタについても解説します．

3.1 CPU近傍のデバイスの概要

この節では，各デバイスの仕様を簡単に説明します．詳しい説明は，次節以降を参照してください．

3.1.1 CPU近傍のデバイスとその仕様

表I-3-1に，FMTOWNSのCPU近傍のデバイスと仕様を示します．

●CPUとNDP(数値演算プロセッサ)

CPUは80386で，クロック周波数は16MHzで動作します．

NDP(数値演算プロセッサ)はオプションで，80387数値演算プロセッサカードによって増設します．

●RAMとROM

RAMは，最大6MBまで増設できます．FMTOWNSの出荷時には，モデル1は1MB，モデル2は2MB実装されています．増設には，拡張RAMモジュールを使用します．1MBのモジュールと，2MBのモジュールの2種類があり，1MBまたは2MB単位で増設できます．

ROM領域には，システム立ち上げ時の起動プログラムやBIOS，漢字ROMなどが含まれます．そのほかに，増設ROMスロットには，システム起動直後に自動実行するプログラムを書いたROMカードが挿入できます．ROMカードが実装されていると，起動直後にはこのROMに制御が渡されます．

●割り込み

15種類の割り込みが使用できます．また，割り込みコントローラとして，8259A相当モジュールが使われています．

なお，強制割り込みのNMIは，通常の割り込みのようにマスク(割り込みを禁止)することはできません．キーボードからのNMIを働かせる際に使用されます．

●DMA

CPUを経由しないバス上のデータ転送をDMA転送といいます．フロッピィディスク，プリンタ，SCSIインタフェース，CD-ROMドライブなどでは，DMA転送が可能です．DMAコントローラには71071が使われています．

●その他

タイマやクロックなどのデバイスや，その他の制御を行うためのレジスタが搭載されています．これらの詳細についても解説します．

▼表I-3-1 CPU近傍の仕様

項目	仕様
CPU	80386 16MHz NDP(80387)使用可能
RAM	容量1MB(モデル1)，2MB(モデル2) 最大6MB(オプションによる増設)
ROM	容量1.5MB
割り込み	8259相当モジュール×2 レベル 用途 0 タイマ 1 キーボード 2 RS-232C 3 拡張RS-232C 4 I/O拡張ユニット 5 I/O拡張ユニット 6 FPD制御 8 SCSI制御 9 CD-ROM* 10 I/O拡張ユニット* 11 VSYNC* 12 プリンタ制御 13 FM，PCM* 14 I/O拡張ユニット 15 予約済

項目	仕様
NMI	RAS機能(キーボード) I/O拡張ユニット*
DMAC	71071 チャネル 用途 0 FPD制御 1 SCSI制御 2 プリンタ制御 3 CD-ROM制御*
拡張DMAC	71071 チャネル 用途 0 拡張スロット* 1 予約済* 2 予約済* 3 予約済*

*のついているものは，FMRと異なる部分

3.2 割り込み

割り込みとは，CPUの仕事中に特別な事象が発生した場合，ハードウェアでこれを検出し，CPUの仕事を中断させて別の仕事をさせることをいいます．

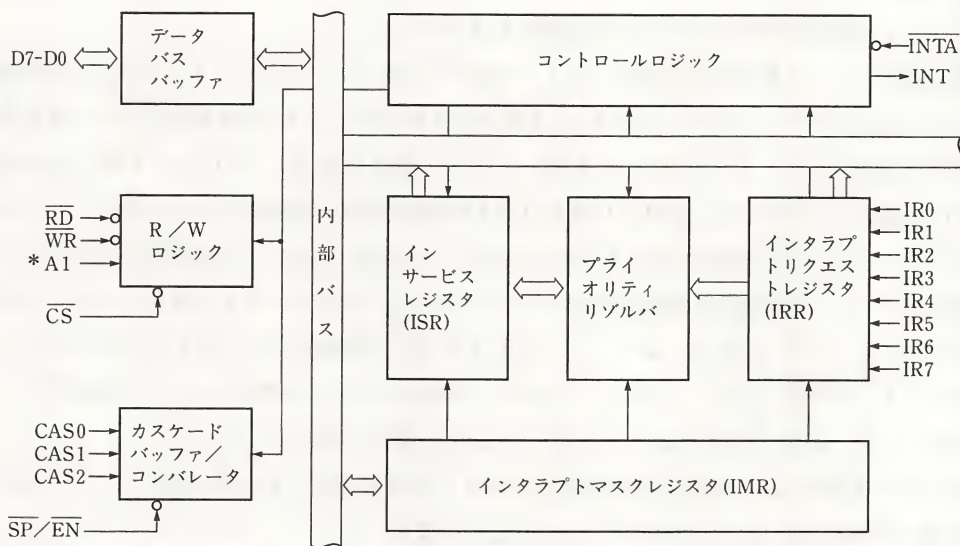
割り込みが起こると，CPUはそれまでの仕事を中断し割り込みハンドラへ制御を移します．

FMTOWNSでは，周辺装置から発生する外部割り込みの制御に，8259A相当のPIC(Programable Interrupt Controller)モジュールが使用されており，15種類の割り込みをサポートしています．

この節では，PICの構造と割り込みの仕組みについて説明します．

3.2.1 PICの構造

8259A相当のPICモジュールの内部ブロック構成を，図I-3-1に示します．

▼図I-3-1　PICのブロック構成

＊A1 PIC本来のピン名はA0であるが，FMTOWNSではA1として使っている．

●割り込みの数

1個のPICには，8本の割り込みライン(IR0－7)があり，8種類の周辺装置からの割り込みを受け付けることができます．

FMTOWNSでは，2個のPICを連結して使用しています．16本の割り込みラインのうち，1本の割り込みラインをPIC同士の連結用に用いるので，割り込みの数は合計15本までとなります．

●割り込みの優先順位

各PICの8本の割り込みラインには，優先順位をつけることができます．また，2個のPICには優先順位があります．優先度の高いPICをマスタ(主)，低いPICをスレーブ(従)と呼びます．こうして，15個の割り込みに優先順位をつけて処理することができます．

割り込みの処理には何とおりかの方法がありますが，各割り込みに優先順位をつけて使うのが標準的であり，これを割り込み優先順モード(フリーネステッドモード)といいます．

PICは，初期化コマンド(ICW，後述)で設定した直後は，このモードになっています．

3.2.2　割り込みの仕組み

PICによる割り込みの制御の仕組みを，最も標準的である割り込み優先順モードの場合について説明します．

PICの内部には，割り込みを制御している重要なレジスタとして，IRR(表Ⅰ-3-2)，ISR(表Ⅰ-3-3)，IMR(表Ⅰ-3-4)があります．

以下に，各レジスタの用途(役割)と，割り込み処理の流れの概要を，ブロック構成図にもとづいて説明します．

割り込みは次のようなメカニズムで処理されます．

周辺装置によって割り込みが発生すると，IRR(インタラプトリクエストレジスタ)のその割り込みに対応するビットが1になります．PICはCPUに対してINT信号を出力し，割り込みの発生を通知します．これに対して，CPUは$\overline{\mathrm{INTA}}$信号を出力し，PICによる割り込み要求(INT)に応答します．すると，PICはCDH(CALL命令の16進の機械語コード)をデータバスに出力します．CPUが引き続き$\overline{\mathrm{INTA}}$信号を出力すると，PICはあらかじめ内部に記憶していた割り込みハンドラ(割り込みを処理するサービスルーチン)のアドレスを，8ビットずつデータバスに出力してCPUに転送します．このとき，PICは，処理中の割り込みを，ISR(インサービスレジスタ)に記憶します．インサービスとは，「現在サービス(処理)中」という意味です．このようにして，CPUは割り込みハンドラへのCALL命令を実行します．

割り込み処理中は，その割り込みの情報はIRRとISRに保存され，割り込みハンドラからのEOI(割り込み終了)コマンドを受けたとき消去されます．

なお，割り込み処理中に他の割り込みが発生すればIRRにそれが記憶されます．この場合，現在，行われている割り込みと，新しい割り込みの優先度が比べられ，新しい割り込みの優先度が高い場合には，現在進行中の割り込み処理を中断して，新しい割り込みの処理を行います．

ただし，割り込み処理中には，他の割り込みを受け付けないようにすることもできます．それには，IMR(インタラプトマスクレジスタ)の各割り込みに対応するビットを1に設定します(フラグを立てる)．

プライオリティリゾルバ(プライオリティ決定回路)は，常時，IRRとISRとIMRの状態を調べ，発生した割り込み(IRR)と処理中の割り込みのレベル(ISR)とマスクの状況(IMR)に従って，処理すべき割り込みを決めています．

▼表 I-3-2　IRR（インタラプトリクエストレジスタ）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0000H	インタラプトリクエストレジスタ　（マスタ側）	R	IR7 (SINT)	IR6 (INT6)	IR5 (INT5)	IR4 (INT4)	IR3 (INT3)	IR2 (INT2)	IR1 (INT1)	IR0 (INT0)

IR7-0(bit7-0)　：SINT，INT6-0から割り込み要求が発生した場合，そのすべての割り込みレベルを該当するビットに記憶する．

SINTはスレーブ側からの割り込みを示す．
INT6-0はハードウェアからの割り込み要因を示す．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0010H	インタラプトリクエストレジスタ　（スレーブ側）	R	IR7 (INT15)	IR6 (INT14)	IR5 (INT13)	IR4 (INT12)	IR3 (INT11)	IR2 (INT10)	IR1 (INT9)	IR0 (INT8)

IR7-0(bit7-0)　：INT15-8から割り込み要求が発生した場合，そのすべての割り込みレベルを該当するビットに記憶する．

INT15-8はハードウェアからの割り込み要因を示す．

▼表 I-3-3　ISR（インサービスレジスタ）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0000H	インサービスレジスタ　（マスタ側）	R	IS7 (SINT)	IS6 (INT6)	IS5 (INT5)	IS4 (INT4)	IS3 (INT3)	IS2 (INT2)	IS1 (INT1)	IS0 (INT0)

IS7-0(bit7-0)　：SINT，INT6-0の内，現在サービス中である割り込みレベルを該当するビットに記憶する．

SINTはスレーブ側からの割り込みを示す．
INT6-0はハードウェアからの割り込み要因を示す．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0010H	インサービスレジスタ　（スレーブ側）	R	IS7 (INT15)	IS6 (INT14)	IS5 (INT13)	IS4 (INT12)	IS3 (INT11)	IS2 (INT10)	IS1 (INT9)	IS0 (INT8)

IS7-0(bit7-0)　：INT15-8の内，現在サービス中である割り込みレベルを該当するビットに記憶する．

INT15-8はハードウェアからの割り込み要因を示す．

▼表 I-3-4　IMR（インタラプトマスクレジスタ）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0002H	インタラプトマスクレジスタ　（マスタ側）	R	IR7	IR6	IR5	IR4	IR3	IR2	IR1	IR0
0012H	インタラプトマスクレジスタ　（スレーブ側）	R	IR15	IR14	IR13	IR12	IR11	IR10	IR9	IR8

IR15-0　：割り込みマスクの状態を示す．
0＝許可
1＝禁止

3.2.3 PICの制御

PICの制御は，前に述べたIRR，ISR，IMRの各レジスタとICW(初期化コマンドワード)，OCW(動作コマンドワード)を使って行います．

IRR，ISR，IMRの各レジスタの内容の読み出し，ICW，OCWの書き込みは，I/Oアドレス上で，CPUから行います．

IMRへの書き出しは，OCWへの書き込みを通じて，間接的に行われます．なお，IRR，ISRの書き出しは，割り込み時と割り込み終了時に行われます．

レジスタの内容の読み出し，コマンドワードの書き込みといった，PICの基本操作には，A1，D4，D3，$\overline{RD}$，$\overline{WR}$，$\overline{CS}$の信号が関係しています．これらの信号と操作の関係を表Ⅰ-3-5に示します．

▼表Ⅰ-3-5　8259Aの基本オペレーション

オペレーションモード	内容	A1	D4	D3	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$
読み出し	IRR，ISR ――→ データバス[*1]	0			0	1	0
	IMR ――→ データバス	1			0	1	0
書き込み	データバス ――→ OCW2	0	0	0	1	0	0
	データバス ――→ OCW3	0	0	1	1	0	0
	データバス ――→ ICW1	0	1	×	1	0	0
	データバス ――→ OCW1，ICW2[*2]，ICW3[*2] ICW4[*2]	1	×	×	1	0	0
ディセーブル	データバス ――→ 切り離し	×	×	×	1	1	0
	データバス ――→ 切り離し	×	×	×	×	×	1

＊1　リードオペレーションに先立ち，IRRとISRのうちどれを読み出すかを，動作コマンドワード3(OCW3)で指定する．
＊2　初期化コマンドワード(ICW)は，PICに内蔵のシーケンスロジックによって，適当な順番で行われる．

A1の値は，I/Oアドレスと関係しています．A1の値とマスタ，スレーブの2つのPICをアクセスする場合のI/Oアドレスの関係は，表Ⅰ-3-6のようになります．

▼表Ⅰ-3-6　I/OアドレスとPICのA1との対応

I/Oアドレス		8259 A1
マスタ	スレーブ	
0000H	0010H	0
0002H	0012H	1

なお，$\overline{CS}$の値は，該当PICをアクセスすると0になります．

● ICWを使ったPICの初期化

PICを使う前には，まず，初期化コマンドワード(ICW)によってPICの初期化をしなければなりません．

初期化コマンドには，ICW1(表I-3-7)，ICW2(表I-3-8)，ICW3(表I-3-9)，ICW4(表I-3-10)があります．

図I-3-2に，それらのICWを使用して初期化を行う手順を示します．FMTOWNSは2個のPICを搭載しているので，ICW1，ICW2，ICW4の外に，ICW3も使用します．

表I-3-10中の高優先度割り込みモード(スペシャルフリーネステッドモード)は，スレーブ側の割り込み処理中に，スレーブ側の別な高優先度割り込みを可能にするモードです．このモードを指定するときはマスタ側のみICW4のSFNMを1にします．また，バッファードモードとはハードウェアのバッファを制御するモードで，FMTOWNSでは必ず1にセットします．

初期状態は，ICW1〜ICW4で設定する値によって変わります．

ICW1〜ICW4は，必ずこの順序で書き込まなければならず，任意のICWだけを書き込むことはできません．したがって，ICWのいずれか1つを書き換えたいときでも，ICW1からICW4までの設定を行う必要があります．

なお，ICW1を書き込んだ時点で，IRRとISRはクリアされ，フリーネステッドモードに初期化されます．このとき，レジスタ読み出しコマンドにおけるIRRとISRの選択はIRRに初期化されます．

▼表I-3-7　ICW1（初期化コマンドワード1）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0000H	初期化コマンドワード1（マスタ側）	W	A7（0）	A6（0）	A5（0）	1	LTIM（1）	ADI（0）	SNGL（0）	IC4（1）
0010H	初期化コマンドワード1（スレーブ側）	W								

A7-5(bit7-5)　：80386の場合無視される．
　　0＝固定

LTIM(bit3)　：IR入力のトリガモードを設定する．
　　1＝レベルトリガモード(固定)

ADI(bit2)　：80386の場合無視される．
　　0＝固定

SNGL(bit1)　：PICを1個使用するか複数個使用する(カスケードモード)かの指定を行う．
　　0＝カスケードモード(固定)

IC4(bit0)　：ICW4が必要か否かの設定を行う．
　　1＝必要(固定)

▼表I-3-8 ICW2(初期化コマンドワード2)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0002H	初期化コマンドワード2(マスタ側)	W	A15/T7	A14/T6	A13/T5	A12/T4	A11/T3	A10(0)	A9(0)	A8(0)
0012H	初期化コマンドワード2(スレーブ側)	W								

A15/T7-A11/T3 (bit7-3) : 割り込みベクタアドレスの設定を行う.

A10-8(bit2-0) : 80386の場合無視される.
0=固定

▼表I-3-9 ICW3(初期化コマンドワード3)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0002H	初期化コマンドワード3(マスタ側)	W	S7(1)	S6(0)	S5(0)	S4(0)	S3(0)	S2(0)	S1(0)	S0(0)

S7-0(bit7-0) : 対応するIR入力(IR0-7)にスレーブを接続するか否かを設定する.
1=スレーブを接続する(固定)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0012H	初期化コマンドワード3(スレーブ側)	W	0	0	0	0	0	ID2(1)	ID1(1)	ID0(1)

A7-5(bit7-3) : 80386の場合無視される.
0=固定

ID2-0(bit2-0) : スレーブ側の識別番号(7を設定).

▼表I-3-10 ICW4(初期化コマンドワード4)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0002H	初期化コマンドワード4(マスタ側)	W	0	0	0	SFNM	BUF(1)	M/S	AEOI	μPM(1)
0012H	初期化コマンドワード4(スレーブ側)	W								

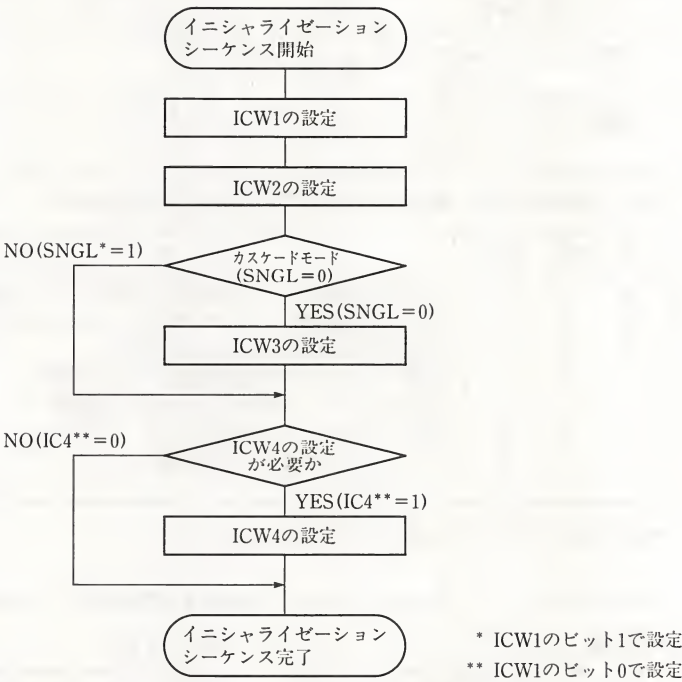
SFNM(bit4) : スペシャルフリーネステッドモードの設定を行う.
0=フリーネステッドモード
1=スペシャルフリーネステッドモード

BUF(bit3) : バッファードモードの設定を行う.
1=バッファードモード(固定)

M/S(bit2) : バッファードモードにおける, マスタかスレーブかの設定を行う.
0=スレーブ
1=マスタ

AEOI(bit1) : 割り込み自動終了モードの設定を行う.
0=自動EOI以外のモード
1=自動EOIモード

μPM(bit0) : CPUモードの設定を行う.
1=80386モード(固定)

▼図 I-3-2　イニシャライゼーションシーケンス

● OCW を使った PIC の動作制御

OCW には，OCW1(表 I-3-11)，OCW2(表 I-3-12)，OCW3(表 I-3-13)の 3 種類があります．OCW1～OCW3 の書き込みは，I/O のアドレスと D3 の信号によって区別し，設定や変更を必要とするものだけを書き込むことができます．なお，OCW の表の説明の中には，これまでに説明していないモードもありますが，それらは後で説明します．

▼表 I-3-11　OCW1（動作コマンドワード 1）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0002H	動作コマンドワード 1 (マスタ側)	W	M7	M6	M5	M4	M3	M2	M1	M0
0012H	動作コマンドワード 1 (スレーブ側)	W	M15	M14	M13	M12	M11	M10	M9	M8

M15-0(bit7-0)　：割り込み要求に対する許可，禁止の設定を行う．
0＝許可
1＝禁止

▼表Ⅰ-3-12　OCW2（動作コマンドワード2）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0000H	動作コマンドワード2 (マスタ側)	W	R	SL	EOI	0	0	L2	L1	L0
0010H	動作コマンドワード2 (スレーブ側)	W								

R, SL, EOI(bit7-5)：優先順位の回転と割り込み終了の組み合せ.

R	SL	EOI	機　能
0	0	1	割り込み優先順モードで非特殊EOIコマンドを使用
0	1	1	割り込み優先順モードで特殊EOIコマンドを使用*
0	0	0	自動EOIにおける回転モードを解除
1	0	1	自動回転モードで非特殊EOIコマンドを使用
1	1	1	自動回転モードで特殊EOIコマンドを使用*
1	0	0	自動回転モードで自動EOIモード
1	1	0	プライオリティコマンドの設定*
0	1	0	ノーオペレーション

＊は，割り込みビットの指定にL2-0を使用する.

L2-0(bit2-0)　　：SL(bit6)が1のときに動作する割り込みレベルの設定を行う.

マスタ側

L2	L1	L0	割り込みレベル
0	0	0	0
0	0	1	1
0	1	0	2
0	1	1	3
1	0	0	4
1	0	1	5
1	1	0	6
1	1	1	7 (SINT)

スレーブ側

L2	L1	L0	割り込みレベル
0	0	0	8
0	0	1	9
0	1	0	10
0	1	1	11
1	0	0	12
1	0	1	13
1	1	0	14
1	1	1	15

SINTはスレーブ側からの割り込みを示す.

▼表 I-3-13　OCW3（動作コマンドワード３）

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0000H	動作コマンドワード３（マスタ側）	W	0	ESMM	SMM	0	1	P	PR	RIS
0010H	動作コマンドワード３（スレーブ側）	W								

ESMM，SMM (bit6, 5)　：２つのビットの組み合わせによって，スペシャルマスクモードのコントロールを行う．

ESMM	SMM	機　能
0	0	何もしない(ESMMが０のため)
0	1	
1	0	スペシャルマスクモードを解除し，ノーマルマスクモードにもどる
1	1	スペシャルマスクモードに設定

P (bit2)　：CPUによるハードウェア割り込みを使用せず，ソフトウェアポーリングによる割り込み処理を行う場合に設定する．
ポールコマンドモードを使用するときには，事前にCPUの割り込みを禁止しておくこと．

０＝ポールコマンドモードを使用しない．
１＝ポールコマンドモードを使用する．

PR，RIS (bit1, 0)　：２つのビットの組み合わせによって，レジスタ読み出しコマンドの設定を行う．

PR	RIS	機　能
0	0	何もしない(PRが０のため)
0	1	
1	0	IRRの読み出しの指定
1	1	ISRの読み出しの指定

3.2.4　割り込み制御モード

割り込みの制御には，これまで，説明してきた割り込み優先順モードを含め，さまざまなモードがあります．

●割り込み優先順モード

PICは，ICWで初期設定した直後には，割り込み優先順モードになっています．このモードでは，IR0が最も優先度が高く，IR1，IR2……と，番号が大きくなるほど優先度が低くなります．

割り込みは，80386CPUにハンドラ対応アドレスが出力される段階で，ISRの該当ビットに記憶されていますが，割り込み処理の終了後には，クリアする（0にする）必要があります．そのためには，割り込みハンドラは，処理の終了時に，マスタとスレーブ両方に対して，OCW2に書き込みを行って，EOIを1にしなければなりません．EOI(End Of Interrupt)のことを割り込み終了コマンドといいます．特殊EOIコマンドと非特殊EOIコマンド(後述)の2種類があり，どちらかを選択できます．また，終了コマンドを必要としない自動EOIモード(後述)にすることもできます．

●自動回転モード

優先度を均等にしたいときには，割り込み優先順モードに代えて，自動回転モードを利用します．このモードでは，受け付けられた割り込みの優先度は，割り込みハンドラの起動後に最低の優先度に下げられます．したがって，他の割り込みが連続して発生する場合には，それらの処理が終了しないとその割り込み処理ができないことになります．

自動回転モードでの割り込み終了時の処理は，EOIコマンド(後述)と，自動EOIモード(後述)による終了のどちらかを選択できます．

●割り込み終了コマンドの種類

EOIコマンドは，OCW2のSLによって，2種類の働きをします．

SLビット	意味
0	非特殊EOIコマンド
1	特殊EOIコマンド

非特殊EOIコマンドでは，最高優先順位の割り込みのISRの該当ビットがクリアされます．

特殊EOIコマンドでは，L0〜L2で指定された，特定のビットがクリアされます．

●自動EOIモード

EOIコマンドの手続きが面倒な場合には，自動EOIモードを利用することができます．

このモードを使用するには，初期化する際に，ICW4のAOEI(ビット1)を1にしておきます．

自動 EOI モードでは，割り込みに対応する ISR のビットがセットされないので，EOI コマンドは不必要です．ただし，割り込みの処理中に再度割り込みを受け付けてしまうことがあるので，タイミングに十分注意しなければなりません．また，このモードは，マスタのみに使用でき，スレーブには使用できません．

なお，OCW2 のプライオリティコマンドは，処理済みの割り込みの優先順位を最低に落すことにより，後続の順位を上げるのに使われます．

●スペシャルマスクモード

IMR は，IRR に対するマスクとして働き，割り込みに対応するビットをマスクすることにより割り込みを起こさないようにするものでした．しかし，このモードでは，ISR に対するマスクとして働きます．すなわち，現在処理中の割り込みに対して，より優先度が低い割り込みが起こった場合にも，後続の割り込みを可能にします．

この機能を使うには，ISR をセットしている割り込みに対応する IMR のビットをセットして，OCW3 でスペシャルマスクモードに移ります．

●その他のモード

OCW3 のポールコマンドモードは，CPU に対する割り込みを使わないときのためのモードで，CPU が PIC の内容を監視し続けなければならないため，FMTOWNS では現実的ではありません．

3.3　DMA 転送

CPU を経由せずに，周辺装置とメモリの間で直接データの転送を行うことを，DMA(Direct Memory Access) といいます．

DMAC(DMA Controller) によるメモリアクセスは，CPU の MOV 命令によるメモリアクセスと同様に，あるデバイス(例えばディスクドライブ)から読み出したデータを，そのままメモリに次々と並べていくといった処理を行うのに使用されます．

同じ処理は，もちろん CPU でも行うことができますが，多量のデータを転送するような単調な処理は DMA を使って行うようにすると，CPU は外の作業をすることができるようになります．また，周辺デバイスの中には，CPU による処理では追いつかないほど高速な転送を行うものがあり，このような場合にも，DMA 転送を使います．

FMTOWNS では，DMAC に71071が使用されています．

この節では，DMAC の構造と DMA 転送の仕組みについて解説します．

3.3.1　DMACの割り当て

71071は，4チャネルのDMAを並行して行うことができます．ただし，任意の一時点についてみれば，同時にバスを使用できるのは，DMACの4つのチャネルとCPUの中のいずれか1つです．なお，各チャネルの割り当ては，表I-3-1に示したとおりです．

71071はいろいろな機能を持っていますが，実際にパソコン上で使用されるときには，CPUの補助的な役割を果たすものとして，機能を制限して使用するように設計されることが多いものです．FMTOWNSでも，71071の機能の中で使用禁止となっている部分があるので注意が必要です．

●動作概略

DMACは，CPUなどのDMA転送要求信号によって動作を開始します．ソースとなるデバイス(例えばディスクドライブ)からデータを1バイト(または1ワード)読み込み，それをデスティネーションとなるデバイス(例えばメモリ)に書き込んで，1回分の処理を終えます．このとき，転送されるデータはCPUを経由しません．

読み書きするメモリの先頭アドレスは，あらかじめDMACアドレスレジスタに設定しておき，DMA転送をする回数を，カウントレジスタに設定しておきます．DMAの4チャネルのうちどのチャネルに関するレジスタの設定を行うかは，チャネルレジスタで選択します．

CPUを使って設定を行うのはここまでです．実際のデータ転送はDMACが，その内部のレジスタの値を使用しながら行います．例えば，2バイトずつ転送する場合には，DMACは内部のアドレスレジスタの値を2ずつ加算しながら，カウントレジスタに設定された回数だけDMA転送を行います．

●拡張DMAC(00B0H～00BFH)

拡張DMACは，I/O拡張ユニットによりサポートされます．

レジスタの割り付けは，00B0H～00BFHで，内部DMACの00A0H～00AFに対応します．

3.3.2 DMACのレジスタ

DMACには，各チャネルに共通したレジスタと，各チャネルごとに設定するレジスタがあります．

各チャネルに共通したレジスタには，次のようなものがあります．

イニシャライズレジスタ……………………DMACの初期化とバスサイズの指定
デバイスコントロールレジスタ…………DMACの動作モードの設定
ステータスレジスタ…………………………DMA要求の有無とDMAの終了の保持
リクエストレジスタ…………………………ソフトウェアによるDMA要求
マスクレジスタ………………………………ハードウェアからのDMA要求のマスク

各チャネルごとに設定するレジスタには，次のようなものがあります．

チャネルレジスタ……………………………アクセスするレジスタの指定
ベース／カレントカウントレジスタ……DMA転送の回数の設定
ベース／カレントアドレスレジスタ……DMA転送アドレスの設定
モードコントロールレジスタ……………チャネルごとのモードの設定

DMACには，上記の他にテンポラリレジスタがありますが，FMTOWNSでは使用できません．

以下に，各レジスタの詳細を説明します．

●イニシャライズレジスタ

DMACを使用する際には，最初にイニシャライズレジスタ（表I-3-14）を設定してから，その他のレジスタを設定します．

イニシャライズレジスタの設定は，DMACの初期化と，バスのアクセスサイズを決める働きをします．このレジスタは8ビットで書き込みを行います．

▼表I-3-14　イニシャライズレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A0H	イニシャライズレジスタ	W	0	0	0	0	0	0	16B	RES

16B(bit1)　：DMACのI/Oバス幅の指定を行う．このビットは，DMACの他のレジスタのアクセスに先立って1にセットしなければならない．
　1＝16ビットバス（固定）

RES(bit0)　：DMACをリセットする．このビットは初期化後自動的にクリアされる．
　1＝リセットON

●チャネルレジスタ

次に，チャネルレジスタ(表Ｉ-3-15)を設定します．これは，以後，どのチャネルのどのレジスタに対してアクセスを行うかを指定するものです．チャネルレジスタは，書き込みと読み出しでフォーマットが異なるので，注意を要します．BASE に関しては，カウントレジスタとアドレスレジスタの項で説明します．

▼表Ｉ-3-15　チャネルレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A1H	チャネルレジスタ	W	0	0	0	0	0	BASE	SELCH	
									SL1	SL0

BASE(bit2)　：アドレス/カウントレジスタのベースレジスタ/カレントレジスタのいずれをアクセスするかを指定する．
0＝リード時ーカレントレジスタ
　ライト時ーベース/カレントレジスタ
1＝ベースレジスタ
通常このビットは0を指定する．

SELCH(bit1-0)　：アクセスするチャネルを指定する．DMACにモード/アドレス/バイト数を設定する場合には，必ず指定しなければならない．

SL1	SL0	チャネル
0	0	チャネル0
0	1	チャネル1
1	0	チャネル2
1	1	チャネル3

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A1H	チャネルレジスタ	R	不定			BASE	SELCH			
							SEL3	SEL2	SEL1	SEL0

BASE(bit2)　：アドレス/カウントレジスタのベースレジスタ/カレントレジスタの現在の選択状態を示す．
0＝リード時ーカレントレジスタ
　ライト時ーベース/カレントレジスタ
1＝ベースレジスタ

SELCH(bit3-0)　：現在選択されているチャネルを示す．ビット3-0チャネルが3～0に対応している．

SELCH	チャネル
SEL 0 = 1	チャネル0
SEL 1 = 1	チャネル1
SEL 2 = 1	チャネル2
SEL 3 = 1	チャネル3

●カウントレジスタ

カウントレジスタ(表Ⅰ-3-16)は，DMA 転送を繰り返す回数を保持するレジスタです．カウントレジスタは，各チャネルごとに，ベースカウントレジスタとカレントカウントレジスタの２本があります．CPU が設定するのは，ベースカウントレジスタで，DMA 転送を繰り返す回数を格納します．カレントカウントレジスタは，おもに DMA 転送中に DMAC が使用するものです．通常の初期化(オートイニシャライズ)時には，ベースカウントレジスタ値がカレントカウントレジスタに転送され，１回 DMA を行うごとに，値が１ずつ減らされ，０に達したときに処理が終了します．CPU から，どちらのカウントレジスタをアクセスするかは，チャネルレジスタの BASE ビットで指定します．

カウントレジスタの長さは２バイト(16ビット)あり，64KB まで連続して DMA 転送できます．

これらのレジスタの内容は，ベース／カレントとも，読み／書き共通です．初期化レジスタで16ビットバスサイズが選択されているときには，２バイトのレジスタを一度にアクセスすることができます．

▼表Ⅰ-3-16　カウントレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A2H	カウントレジスタ(下位)	R/W	C7	C6	C5	C4	C3	C2	C1	C0
00A3H	カウントレジスタ(上位)	R/W	C15	C14	C13	C12	C11	C10	C9	C8

C15-0　：DMA転送を行うバイト数を指定する．

●アドレスレジスタ

アドレスレジスタ(表I-3-17)は，DMA転送をするメモリのアドレスを保持します．アドレスレジスタは，各チャネルごとに，ベースアドレスレジスタとカレントアドレスレジスタの2本があります．CPUが設定するのは，ベースアドレスレジスタで，DMA転送でデータを読み書きする先頭のアドレス(開始アドレス)を格納します．カレントアドレスレジスタは，おもにDMA転送中にDMACが使用するもので，現在，読み書きしているアドレスを保持しています．通常の初期化(オートイニシャライズ)時には，ベースアドレスレジスタ値がカレントアドレスレジスタに転送され，1回DMAを行うごとに，±1(8ビット時)または±2(16ビット時)ずつ自動的に増減されます．なお，ここでの8ビット，16ビットとは，バスサイズではなく，後述のモードコントロールレジスタで指定するデータ幅です．CPUからどちらのアドレスレジスタをアクセスするかは，チャネルレジスタのBASEビットで指定します．

アドレスレジスタの長さは4バイト(32ビット)で，4GB中の任意のアドレスを指定できます．アドレスレジスタの下位3バイトはDMACに内蔵されており，最上位バイトは外付けレジスタです．下位3バイトは連続して増減されますが，A23からA24への桁上がりは行われません．したがって，16MB以上の転送にはアドレスの桁上がり(下がり)は，ソフトウェアで行う必要があります．レジスタの内容は，ベース／カレントとも，読み／書き共通です．

また，最上位バイトのアドレスレジスタは，1つしかなく，ベースとカレントで兼用しているので注意が必要です．

▼表I-3-17　アドレスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A4H	アドレスレジスタ(下位)	R/W	A7	A6	A5	A4	A3	A2	A1	A0
00A5H	アドレスレジスタ(中位)	R/W	A15	A14	A13	A12	A11	A10	A9	A8
00A6H	アドレスレジスタ(上位)	R/W	A23	A22	A21	A20	A19	A18	A17	A16
00A7H	アドレスレジスタ(最上位)	R/W	A31	A30	A29	A28	A27	A26	A25	A24

A31-0　：DMA転送の開始アドレス(4GB空間)を指定する．
A31-24は外付けレジスタで，A23からA24への桁上がりは行われない．
したがって，16MB境界をまたいでの転送は回り込みを起こすので，注意すること．

●デバイスコントロールレジスタ

デバイスコントロールレジスタ(表I-3-18)は，各チャネルに共通したデバイス全体のモードの設定を行うレジスタです．

FMTOWNSでは，設計上の制約から，このレジスタの多くのビット値は固定されています．設定が可能なビットはDDMAだけで，このビットはDMA動作の許可／禁止を制御します．

▼表I-3-18　デバイスコントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A8H	デバイスコントロールレジスタ	R/W	AKL (0)	RQL (0)	EXW (1)	ROT (0)	CMP (0)	DDMA	AHLD (0)	MTM (0)

AKL (bit7)　：DMAAKのアクティブレベルを指定する．
0＝アクティブロー(固定)

RQL (bit6)　：DMARQのアクティブレベルを指定する．
0＝アクティブハイ(固定)

EXW (bit5)　：書き込みの場合のモードを指定する．
1＝拡張書き込み(固定)

ROT (bit4)　：優先順位の制御方法を指定する．
0＝固定優先順位(固定)

CMP (bit3)　：DMAサイクルのタイミング制御の方法を指定する．
0＝通常タイミング(固定)

DDMA (bit2)　：DMA動作を禁止する．
0＝DMA動作許可
1＝DMA動作禁止

AHLD (bit1)　：メモリ-メモリ転送の場合にチャネル0のアドレスを固定とする．メモリ-メモリ転送は使用禁止のため，このビットは0に固定する．

MTM (bit0)　：メモリ-メモリ転送を許可するかどうかを指定する．
0＝メモリ-メモリ転送禁止(固定)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A9H	デバイスコントロールレジスタ	R	不定						WEV (0)	BHLD (0)
		W	0	0	0	0	0	0		

WEV (bit1)　：ベリファイ転送時のWAITの許可を指定する．
0＝ベリファイ転送時のWAIT禁止(固定)

BHLD (bit0)　：DMA転送バスモードを指定する．
0＝バスリリースモード(固定)

●モードコントロールレジスタ

モードコントロールレジスタ(表Ⅰ-3-19)は，チャネルごとに動作モードを指定するレジスタです．

モードコントロールレジスタ中のTMODEは，デマンドモードとシングルモードを切り換えるものです．

デマンドモードでは，DMACがDMA要求を受け続けている間，連続して繰り返しDMA転送を行い，転送が終了するまでバスを解放しません．一方，シングルモードでは，1バイト(または2バイト)単位でバスを解放しながらDMA転送を行います．

▼表Ⅰ-3-19　モードコントロールレジスタ

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="4">00AAH</td><td rowspan="4">モードコントロールレジスタ</td><td rowspan="2">R</td><td colspan="2">TMODE</td><td rowspan="4">ADIR</td><td rowspan="4">AUTI</td><td colspan="2">TDIR</td><td rowspan="2">不定</td><td rowspan="4">W/B</td></tr>
<tr><td>MD1</td><td>MD0</td><td>DIR1</td><td>DIR2</td></tr>
<tr><td rowspan="2">W</td><td colspan="2">TMODE</td><td colspan="2">TDIR</td><td rowspan="2">0</td></tr>
<tr><td>MD1</td><td>MD0</td><td>DIR1</td><td>DIR2</td></tr>
</table>

TMODE(bit7-6)　：DMA転送モードを指定する．

MD1	MD0	転送モード
0	0	デマンドモード
0	1	シングルモード
1	0	ブロックモード(使用禁止)
1	1	カスケードモード(使用禁止)

ADIR(bit5)　：アドレスカウンタのインクリメント/デクリメントを指定する．
0＝アドレスインクリメント
1＝アドレスデクリメント

AUTI(bit4)　：オートイニシャライズを行うかどうかを指定する．
0＝オートイニシャライズを行わない．
1＝オートイニシャライズを行う．

TDIR(bit3-2)　：DMA転送の転送方向を指定する．

DIR1	DIR0	転送モード
0	0	ベリファイ転送(使用禁止)
0	1	I/O　⇨ メモリ転送
1	0	メモリ ⇨ I/O　転送
1	1	メモリ ⇨ メモリ転送(使用禁止)

W/B(bit0)　：DMA転送のデータ幅を指定する．
0＝バイト転送
1＝ワード転送

●ステータスレジスタ

ステータスレジスタ(表I-3-20)は，DMA要求信号の有無と転送が終了しているかどうかを保持するレジスタです．

▼表I-3-20　ステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00ABH	ステータスレジスタ	R	REQUEST				TERMINAL COUNT			
			RQ3	RQ2	RQ1	RQ0	TC3	TC2	TC1	TC0

RQ3-0(bit7-4)　：DMA要求信号の状態を示す．RQ3-0が，チャネル3～0に対応する．
0＝DMA要求なし
1＝DMA要求あり

TC3-0(bit3-0)　：DMA転送が指定されたバイト数終了したかどうかを示す．このビットはリードするとクリアされる．TC3-0がチャネル3～0に対応する．
1＝ターミナル　カウント状態．

●テンポラリレジスタ

テンポラリレジスタ(表I-3-21)は，メモリからメモリへの転送の中継に使用されますが，FM TOWNSではこの機能は使っていません．

▼表I-3-21　テンポラリレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00ACH	テンポラリレジスタ(下位)	R	T7	T6	T5	T4	T3	T2	T1	T0
00ADH	テンポラリレジスタ(上位)	R	T15	T14	T13	T12	T11	T10	T9	T8

このレジスタは使用しない．

●リクエストレジスタ

リクエストレジスタ(表I-3-22)は，ソフトウェアからDMA要求を発するためのレジスタです．すなわち，ソフトウェアがこのレジスタ中のチャネルに対応したビットをセットすると，DMACにそのチャネルのDMA要求が伝わります．

▼表I-3-22　リクエストレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00AEH	リクエストレジスタ	R	不　定				SRQ3	SRQ2	SRQ1	SRQ0
		W	0	0	0	0				

SRQ3-0(bit3-0)　：ソフトウェアによるDMA要求．SRQ3-0がチャネル3～0に対応．
0＝DMA要求リセット
1＝DMA要求セット

●マスクレジスタ

マスクレジスタ(表Ⅰ-3-23)は，ハードウェアからのDMA要求を受け付けるか否かを設定するレジスタです．各チャネルに対応するビットをセットすると，そのチャネルのハードウェアからのDMA要求信号がマスクされます．

なお，DMACのセットに際しては，該当チャネルについてマスクしておいて行わないと誤動作するおそれがあります．

▼表Ⅰ-3-23 マスクレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00AFH	マスクレジスタ	R	不定				M3	M2	M1	M0
		W	0	0	0	0				

M3-0(bit3-0)　：DMA要求信号のマスクビットであり，M3-0がチャネル3～0に対応する．
0＝DMA要求をマスクしない
1＝DMA要求信号をマスクする

3.4 プログラマブルタイマ

プログラマブルタイマ(PIT)は，入力された一定の周波数のクロックパルスを数えるカウンタで，カウント値がタイムアウトする(指定値に達する)と，割り込みを発生したり，パルスを出力する働きをします．

PITは，一定の時間間隔で割り込みを発生させることを利用して，インターバルタイマとして使うことができます．ソフトウェアは，この割り込みを利用して，タスクの切り換えなどを行うことができます．また，一定の時間間隔でパルスを発生することを利用して，RS-232Cのボーレートの制御では，ボーレートジェネレータとなります．

FMTOWNSでは，PITに，8253相当のモジュールを使用しています．

この節では，8253の構造とその働きの仕組みについて，解説します．

3.4.1 タイマの割り当てと注意

1個の8253には，3チャネルの独立したタイマ(カウンタ)が内蔵されており，FMTOWNSでは，このPITを2個使用しています．ただし，チャネル3，5は予約されているので，使用できるのは4チャネルです．各チャネルの用途と入力クロックの周波数を，表Ⅰ-3-24に示します．

▼表 I-3-24　タイマ各チャネルの仕様

項　目	仕　　様
タ イ マ	8253×2
チャネル割り当て	チャネル　用途 0　ソフト(インターバルタイマ) 1　I/O制御用 2　サウンド 3　予約済 4　ボーレートジェネレータ 5　予約済
カウントクロック	チャネル　カウントクロック 0　307.2KHz 1　307.2KHz 2　307.2KHz 3　―――― 4　1.2288MHz 5　――――
割り込み	チャネル０，１のみタイムアウトによる割り込み可能.

FMTOWNS では，8253の動作モードのうち，チャネル０，２，３ではモード３，チャネル１はモード０を使用しています．このモードの設定は，設計上固定されているので，変更することはできません．

また，割り込みの発生に使用できるのは，チャネルの０，１だけです．

PIT1，PIT2 を連続してアクセスするときは，1.3μs 以上，間をおいてください．

いずれのタイマもゲート入力によるカウント制御はできません．

チャネル０では，設定した周期で割り込み要因レジスタ(後述)のタイムアウトフラグ(TMOUT0)がセットされるので割り込み処理ルーチンでは，このフラグをリセットします．すなわち，割り込み制御レジスタ(後述)の TM0CLR に１を書くようにしてください．

3.4.2 PIT のレジスタ

PIT には，次のようなレジスタがあります．

タイマカウントレジスタ……………………カウント値(時間)の設定
コントロールレジスタ………………………PIT の動作モードを指定
割り込み制御レジスタ………………………割り込みの発生の許可／禁止
割り込み要因レジスタ………………………割り込みの状況

次に，各レジスタの詳細を説明します．

●タイマカウントレジスタ

タイマカウントレジスタ(表I-3-25)には，各チャネルごとに，入力クロックのカウント値を設定します．タイマカウントレジスタは，それぞれ16ビットです．

このレジスタに設定した値までカウントが進むとタイムアウトになり，割り込みなどが発生します．

チャネル4は，RS-232C専用のボーレートジェネレータで，8251の分周比とボーレートによって設定値が決まっています．チャネル4のタイマカウントレジスタの設定値を，表I-3-26に示します．なお，RS-232Cに関しては，第I部第7章に詳しい解説があります．

▼表I-3-25　タイマカウントレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0	
0040H	タイマカウントレジスタ	R/W	タイマ　#0								PIT1
0042H		R/W	タイマ　#1								
0044H		R/W	タイマ　#2								
0050H		R/W	タイマ　#3								PIT2
0052H		R/W	タイマ　#4								
0054H		R/W	タイマ　#5								

0～5のチャネルのタイマにカウント値を設定する．

▼表I-3-26　ボーレートジェネレータに設定するカウント値

モード	同　期	非　同　期		カウント値
分周比	1/1	1/16	1/64	
ボーレート	75			16384
	150			8192
	300			4096
	600			2048
	1200	75		1024
	2400	150		512
	4800	300	75	256
	9600	600	150	128
	19200	1200	300	64
		2400	600	32
		4800	1200	16
		9600	2400	8
		19200	4800	4

●コントロールレジスタ

コントロールレジスタ(表I-3-27)は，各チャネルに対して，動作モードなどを設定するものです．

チャネルの指定は，SC1-0の2ビットで行いますが，2個のPITは，I/Oアドレスが異なっているので注意してください．チャネル0～2はI/Oアドレス0046Hでアクセスし，チャネル3～5はI/Oアドレス0056Hでアクセスします．

RL1-0のビットは，タイマカウント値を読み出す際の動作およびカウンタラッチ(カウント値の保持)を設定します．タイマカウントは2バイトなので，同じI/Oアドレスから，上位バイトのみ，下位バイトのみ，または連続して読み込みます．

▼表I-3-27　コントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0046H	コントロールレジスタ PIT1　(#0～#2)	W	SC1	SC0	RL1	RL0	M2	M1	M0	BCD
0056H	コントロールレジスタ PIT2　(#3～#5)	W								

SC1-0(bit7-6)　：タイマを選択する．

I/Oアドレス	SC1	SC0	チャネル
0046H (PIT1)	0	0	0
	0	1	1
	1	0	2
0056H (PIT2)	0	0	3
	0	1	4
	1	0	5

RL1-0(bit5-4)　：カウントレジスタにロードするカウント数，およびリードするカウントデータのバイト長またはカウンタラッチ動作の指定を行う．

RL1	RL0	機　能
0	0	カウンタラッチ
0	1	下位バイトリード/ライト
1	0	上位バイトリード/ライト
1	1	下位バイト，上位バイト連続リード/ライト

M2-0(bit3-1)　：タイマの動作モードを指定する．

M2	M1	M0	モード設定
0	0	0	モード0
0	0	1	モード1
×	1	0	モード2
×	1	1	モード3
1	0	0	モード4
1	0	1	モード5

BCD(bit0)　：カウント形式を指定する．
0＝バイナリカウント(16桁)
1＝BCDカウント(4桁)

M2-0はタイマの動作モードの指定です．FMTOWNSでは，チャネルによってモードが固定されているので，必ずこの値を指定します．各モードの詳細については長くなるので，本書では省略します．詳しい解説が必要な場合は，8253のマニュアル等を参照してください．

BCDビットは，カウント形式を指定します．0をセットするとバイナリ(2進)カウントを行い，1をセットすると2進化10進(BCD)でカウントします．BCDでは，4ビット単位に10進の1桁に対応します．BCDカウントは，バイナリカウントに比べて，最大カウント数が小さい点に注意が必要です．

●割り込み制御レジスタ

割り込み制御レジスタ(表I-3-28)には，チャネル0と1について，タイムアウト時の割り込みの禁止／許可の外に，タイマ#0について割り込み要因レジスタのタイムアウトフラグをクリアする機能(TM0CLR)があります．

TM0CLRは，タイムアウトフラグをクリアする際に1を書き込みますが，クリア後は，このビットも消されてしまいます．したがって，ソフトウェアで0にする必要はありません．

また，このレジスタのSOUNDは，サウンド出力のON/OFFに使用されています．

▼表I-3-28　割り込み制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0060H	割り込み制御レジスタ	W	TM0 CLR	0	0	0	0	SOUND	TM1 MSK	TM0 MSK

TM0CLR(bit7)　：タイマ#0のタイムアウトフラグ(TMOUT0)をクリアする．
1＝TMOUT0を0とする．(その後TM0CLRを0にもどす必要はない)

SOUND(bit2)　：サウンド出力を制御する．
0＝サウンド出力をOFFにする
1＝サウンド出力をONにする

TM1MSK(bit1)　：タイマ#1のタイムアウトによる割り込みを制御する．
0＝割り込み禁止
1＝割り込み許可

TM0MSK(bit0)　：タイマ#0のタイムアウトによる割り込みを制御する．
0＝割り込み禁止
1＝割り込み許可

●割り込み要因レジスタ

割り込み要因レジスタ(表 I-3-29)は，割り込み制御レジスタの設定状況と，タイムアウトの状況を示します．

SOUND，TM1MSK，TM0MSK は，割り込み制御レジスタの設定状況を反映しています．

TMOUT1，TMOUT0 は，タイマ#1，０のタイムアウト状態を反映しています．これらのビットは，いずれも１のときタイムアウトが発生したことを示します．

これらのビットをクリアする方法は，次のとおりです．

TMOUT0 は，前述のように TM0CLR ビットに１を書き込みます．

TMOUT1 はタイマカウントレジスタ#０にカウント値を書き込むことによって，自動的にクリアされます．

これは，インターバルタイマ(チャネル０)が一定間隔で動作するのに対し，I/O 制御のタイムアウト(チャネル１)は，デバイスによって設定時間が異なり，その都度，タイマカウントレジスタの書き変えを必要とするためです．

▼表 I-3-29　割り込み要因レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0060H	割り込み要因レジスタ	R	不　定			SOUND	TM1 MSK	TM0 MSK	TM OUT1	TM OUT0

SOUND(bit4)　：割り込み制御レジスタのSOUNDビットの状態を示す．

TM1MSK(bit3)　：割り込み制御レジスタのTM1MSKビットの状態を示す．

TM0MSK(bit2)　：割り込み制御レジスタのTM0MSKビットの状態を示す．

TMOUT1(bit1)　：タイマ#１のタイムアウトを示す．
　０＝カウント中
　１＝タイムアウト
TMOUT1＝１かつTM1MSK＝１ならば，CPUに対し割り込みをかける．
カウント数のセットにより，このビットは０になる．

TMOUT0(bit0)　：タイマ#０がタイムアウトになったことを示す．
　０＝カウント中
　１＝タイムアウト
TMOUT0＝１かつTM0MSK＝１ならば，CPUに対し割り込みをかける．
TM0CLRに１を書くと，このビットは０になる．

3.5 リアルタイムクロック

リアルタイムクロック(RTC)は，年月日と時分秒の両方を管理する「時計」です．この節では，リアルタイムクロックの構造と働きの仕組みについて解説します．

3.5.1 リアルタイムクロックの仕様

リアルタイムクロックの仕様を，表I-3-30に示します．

FMTOWNSのRTCには，従来のFMシリーズの各機種と同じ58321Bが使用されています．NiCd電池によってバックアップされており，フル充電では約3カ月間バックアップされます．

▼表I-3-30　リアルタイムクロックの仕様

項　目	仕　　様
LSI	RTC58321B
バッテリバックアップ	NiCdバッテリにより可能 フル充電時　3ヶ月間バックアップ可能
表示データ	毎月日時分秒

3.5.2 RTC内部のレジスタ

RTCの内部には，アドレスレジスタと，数多くの4ビットのカウンタがあります．アドレスレジスタは，CPUからカウンタをアクセスする際に，使用されます．

カウンタは，時刻値に対応するもので，1秒の位，10秒の位，……，1年の位，10年の位を保持しています．これらのカウンタの値は，時間が進むにしたがって，刻々と変わります．データレジスタへの書き込みは，年月日や時分秒の値のセット，読み出しは現在値を取得することになります．その内容を表I-3-31に示します．

RTCでは，水晶発振による原クロックの周波数を分周回路で数えながら，1秒ごとのクロックパルスを生成しています．

この分周回路とカウンタの関係を，図I-3-3に示します．秒クロックは，1秒のレジスタと連結していますが，1秒の位の繰り上がりが，分，時，日，曜，月，年のレジスタに影響を及ぼすことがあります．RTCは，必要に応じて各レジスタの内容を書き換えるために，1秒のクロックの発生直後は，CPUからレジスタに書き込みを行うことはできません．

RTCは，1秒のクロックに合わせて，図I-3-4のようなビジー信号を出しています．RTCレジスタに書き込みを行うときは，ビジー状態でないことを確かめた上で書き込みを開始します．また，書き込み中にビジー状態になったときには，書き込み失敗と見なして再度書き込みを行う必要があります．

時刻データの書き込み中にビジーが発生した場合に，厳密さを要求するなら，再び1秒の位からセットし直すべきです．例えば，「29秒」を設定するために，「9」を書き込んだ直後にビジーが発生すると，1秒の位は「0」になってしまい，そのまま「2」を書き込むと，RTCの内容は「20」秒になってしまいます．

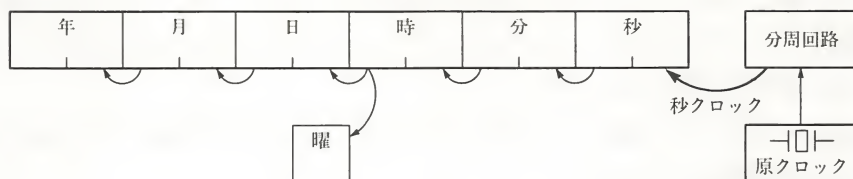
時報に合わせて秒の下位の値を0にするといった操作であれば(現実的にはこのような操作が多い)，あまり支障はありません．

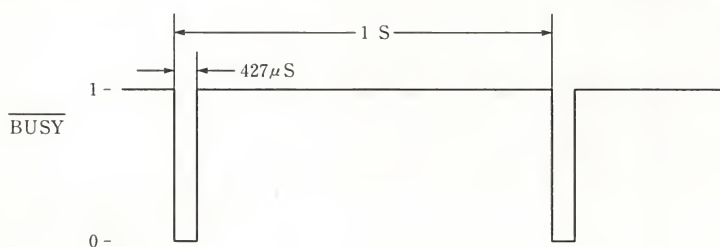
▼表 I-3-31　RTC内部のデータレジスター覧表

内部カウンタ	アドレス	ビット表現				データ				カウント値	備考
		D_3	D_2	D_1	D_0	D_3	D_2	D_1	D_0		
S1(秒)	0	0	0	0	0	*	*	*	*	0～9	
S10(10秒)	1	0	0	0	1		*	*	*	0～5	
M11(分)	2	0	0	1	0	*	*	*	*	0～9	
M110(10分)	3	0	0	1	1		*	*	*	0～5	
H1(時)	4	0	1	0	0	*	*	*	*	0～9	
H10(10時)	5	0	1	0	1	*[1]	*	*	*	0～1 / 0～2	D_2＝1にてPM，D_2＝0でAM，D_3＝1にて24H計時，D_3＝0で12H計時，D_3＝1をWRITEするとD_2のビットはIC内部でリセットされ常に0となる．
W(曜)	6	0	1	1	0		*	*	*	0～6	
D1(日)	7	0	1	1	1	*	*	*	*	0～9	
D10(10日)	8	1	0	0	0	*[2]	*[2]	*	*	0～3	D10桁のD_3とD_2ビットは閏年のセレクト用ビット（下表参照）
M01(月)	9	1	0	0	1	*	*	*	*	0～9	
M010(10月)	A	1	0	1	0				*	0～1	
Y1(年)	B	1	0	1	1	*	*	*	*	0～9	
Y10(10年)	C	1	1	0	0	*	*	*	*	0～9	
	D	1	1	0	1						$1/2^{15}$分周段後5段分とBUSY回路をリセットするためのセレクト，アドレスレジスタにこのコードをラッチし，WRITEを1にするとリセットがかかる．
	E～F	1	1	1	0/1						基準信号を得るためのセレクト，アドレスレジスタにこのコードをラッチしREADを1にすると，D_0～D_3に基準信号が出力される．

暦	D_3	D_2	年を4で割った端数
西暦/平成暦	0	0	0
昭和暦	0	1	3
予備	1	0	2
予備	1	1	1

注）・データ入力の空欄は対応ビットなし，READを行うと0レベルが出力され，WRITEを行うとビットがないので記憶されない．
・*[1]印のビットは12H/24Hセレクト用，*[2]印のビットは閏年のセレクト用ビット，この3ビットについてもREAD/WRITEが可能．
・アドレス入力はD_0～D_2がバスラインに信号を入れてADDRESS・WRITEを入れるとアドレスレジスタにアドレス情報がラッチされる．

▼図I-3-3　分周回路とカウンタの関係

秒の位に桁上がりが発生すると，上位の項目のすべてに波及する可能性がある．

▼図I-3-4　RTCのビジー信号

0のときビジーを表す．

3.5.3　閏年の選択

RTCには，閏年に対応できるように，4年に1度，2月29日を表示する機能が用意されています．閏年か否かの判断は，年の数字を4で割った剰余(端数)で行います．0～3のどの端数のときに閏年とするかは，日の10の位のレジスタのD2，D3(これらのビットは空いている)で指定します．この方法により，西暦や元号にも柔軟に対応しています．

「西暦」および元号「平成」を使用するときは，年の剰余が0の年が閏年なので，00を指定します．例えば，西暦1992年，1996年……や，平成4，8年などは閏年です．元号「昭和」を使用するときには，剰余が3の年が閏年なので10を指定すればよいことになります．ただし，400年に3回の平年調整については，考慮されていないので注意が必要です．

3.5.4 RTCのレジスタの操作

CPUからRTCの内部レジスタをアクセスするときには，RTCデータレジスタ(表I-3-32)と，RTCコマンドレジスタ(表I-3-33)を使用します．

まず，RTCデータレジスタにアクセスしたい内部レジスタのアドレス(番号)を書き込み，次にその内部レジスタをアクセスします．このとき，RTCコマンドレジスタで，リード／ライトなどの選択をします．

▼表I-3-32　RTCデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0070H	RTCデータレジスタ	R	READY	不　定			D3	D2	D1	D0
		W	0	0	0	0				

READY(bit7)　：RTCが時刻の更新を行っているときにこのビットが0になる．
時刻の更新は1秒ごとに行われ，この間(約430μs)はRTCへの読み書きはできない．
RTCレジスタの読み書きはREADYが1であることを確認してから244μs以内に行う必要がある．244μs以内に終了しない場合は，再度READYが1になったことを確認してから，読み書きを行う．

D3-0(bit3-0)　：RTCへ対してのレジスタ番号，時刻データをセットする．(Write時)
RTCから時刻データを読み出す．(Read時)

▼表I-3-33　RTCコマンドレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0080H	RTCコマンドレジスタ	W	CHIP SELECT	0	0	0	0	READ	WRITE	ADRS WRITE

CHIP SELECT(bit7)：RTCのレジスタの読み書きおよびレジスタ番号を指定するときには，このビットを1にする．このビットを0にすると，他のコマンドは無効になる．

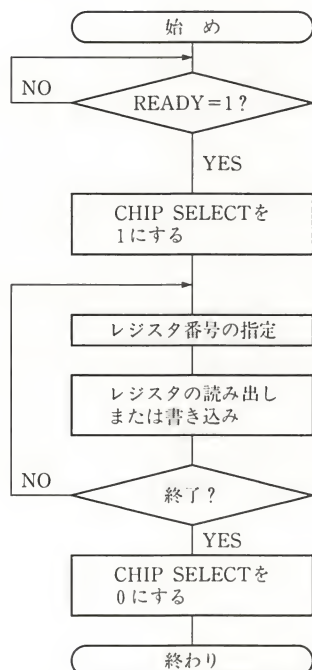
READ(bit2)　：ADRS WRITEにて指定されているRTCレジスタよりデータを読み込む．
このビットを1にしてデータを読み込み，再びこのビットを0にもどすこと．

WRITE(bit1)　：ADRS WRITEにて指定されているRTCレジスタにデータを書き込む．
このビットを1にして，再び0にもどすことによりADRS WRITEにて指定されているRTCレジスタにデータを書き込む．

ADRS WRITE(bit0)：RTCのレジスタの指定を行う．このビットを1にして再び0にもどすことにより，あらかじめデータレジスタに書き込まれているレジスタ番号のRTCレジスタが指定される．

RTCの内部レジスタをアクセスする手順のフローチャートを，図I-3-5に示します．
以下に，アクセスする手順を，フローチャートに沿って説明します．

①RTCのレディ状態を，RTCデータレジスタのビット7で確かめます．このビットが1であれば，RTCはアクセスできる状態です．
②RTCコマンドレジスタに80Hを書き込みます．
③RTCデータレジスタに，RTCの内部レジスタの番号(アドレス)を書き込みます．
RTCコマンドレジスタに81H(ADRS WRITE)を書き込み(RTC内部アドレスレジスタの設定)，続けて80Hを書き込みます．
④RTC内部のレジスタをアクセスします．
RTCの内部レジスタのデータを読み出すときは，RTCコマンドレジスタに84Hを書き込み，2マイクロ秒以上待ってからRTCデータレジスタを読み出します．さらに，RTCコマンドレジスタに00Hを書き込みます．
RTCの内部レジスタにデータを書き込むときは，RTCデータレジスタに，設定したい値を書き込み，RTCコマンドレジスタに82Hを書き込み，2マイクロ秒以上待ってから，RTCコマンドレジスタに80Hを書き込みます．
⑤複数のRTC内部のレジスタをアクセスする場合は③〜④の処理を繰り返し，終了であればRTCコマンドレジスタに00Hを書き込みます．

▼図I-3-5　RTCレジスタの読み書き

3.5.5 分周回路のリセット

RTC内部のレジスタの13(0DH)番に書き込みを行うと(データは何でもよい)，分周回路がリセットされて0にもどります．したがって，その後，約1秒間は，RTCがレディ状態になることはありません．RTCにデータを書き込む際には，この方法を利用すると書き込みの失敗を避けることができます．

3.6 その他のCPU近傍のレジスタ

この節では，CPU近傍に配置されている，補助的なレジスタについて説明します．これらのレジスタは，既存のデバイスとは異なり，独自に設計されたものです．

●リセット要因レジスタ

リセット要因レジスタ(表I-3-34)は，リセットが発生したときに，その原因を示すものです．システムソフトウェアの起動直後に参照して，起動時の処理に反映させるのに使用します．

なお，本体の電源投入時，またはリセットスイッチを押してシステムリセットをしたときには，このレジスタの各ビットは0になります．

▼表I-3-34　リセット要因レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0020H	リセット要因レジスタ	R	不定						SHUT DOWN	SOFT

SHUTDOWN(bit1)　：シャットダウン(CPUによる異常検出)によるリセットが発生したことを示す．このビットはリードすることによりオフにされる．
　1＝シャットダウンリセット

SOFT(bit0)　：ソフトウェアによるリセットが発生したことを示す．このビットはリードすることによりオフにされる．
　1＝ソフトウェアリセット

シャットダウンリセット，ソフトウェアリセットともにCPUとNDPにのみリセットがかかる．
パワーオンリセットおよびシステムリセット時はいずれのフラグも0になる．

●ソフトリセット，NMI ベクタプロテクト，ソフト電源制御レジスタ

ソフトリセット，NMI ベクタプロテクト，ソフト電源制御レジスタ(表Ⅰ-3-35)の RST は，ソフトリセットの指定で，このビットを1にすると，CPU と数値演算プロセッサ(NDP)がリセットされます．その後，リセットシーケンスが開始されるため，誤動作を防ぐために，RST ビットをセットした後は HALT 命令で CPU 待機状態にしておくことが望まれます．

WRPROT は，メモリの NMI ベクタテーブルの内容にライトプロテクトをかけるときに，1にするビットです．

▼表Ⅰ-3-35　ソフトリセット，NMIベクタプロテクト，ソフト電源制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0020H	ソフトリセット，NMIベクタプロテクト，ソフト電源制御	W	WR PROT	POW OFF	0	0	0	0	0	RST

WRPROT(bit7)　：NMIベクタが格納されているメモリアドレスにライトプロテクトをかける．
パワーオンリセットおよびシステムリセット時は0になっている．
0＝NMIベクタライトイネーブル
1＝NMIベクタライトプロテクト

POWOFF(bit6)　：ソフトウェアで電源をOFFにする．
1＝電源OFF

RST(bit0)　：ソフトウェアにより，CPUとNDPにリセットをかける．
1＝ソフトウェアリセットON

DMA転送中にリセットをかけてはならない．
リセット後，必ず0をライトすること．
リセットする際には，事前にPICの割り込みを禁止しておき，リセット後ただちにHALT命令を実行すること．

●電源制御レジスタ

電源制御レジスタ(表Ⅰ-3-36)は，ソフトウェアで電源をオフにするためのレジスタです．電源をオフにしたいときは，POWOFF ビットに1を書き込みます．

▼表Ⅰ-3-36　電源制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0022H	電源制御レジスタ	W	0	POW OFF	0	0	0	0	0	0

POWOFF(bit6)　：ソフトウェアで電源をOFFにする．
1＝電源OFF

● CPU識別レジスタ

CPU識別レジスタ(表I-3-37)は，マシンの機種とCPUの種類を格納しています．

ここで，MACHINE-IDは機種を表わす13ビットのコード，CPU-IDはCPUの種類を表す3ビットのコードです．

FMTOWNSの場合機種の判定に当たっては，プログラムは最初にID7-3が0であることを確かめ，続いてID15-9が0，ID8が1であることを調べます．さきにID15-8を調べると，FMRの場合には不定になっているので，誤った結果が得られることがあります．

▼表I-3-37　CPU識別レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID (bit15-3)　：装置の種別を示す．下記のビット構成により識別を行う．

装　置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不　定								1	1	1	1	1
FMR-50S	不　定								1	1	1	0	1
FMR-70	不　定								1	1	1	1	0
FMTOWNS	0	0	0	0	0	0	0	1	0	0	0	0	0

FMTOWNSのMACHINE-IDは，ID15-8を使用し，ID7-3が0のとき有効である．

CPU-ID(bit2-0)　：使用CPUの種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386
0	1	0	予約済
0	1	1	予約済
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

●シリアルROM制御レジスタ

シリアルROMには機種名，製造番号などが書かれています．このシリアルROM制御レジスタ(表I-3-38)は，書き込みで読み出し制御を行い，読み出しでIDDATAビットから1ビットずつデータが取り出せます．

▼表I-3-38　シリアルROM制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0032H	シリアルROM制御レジスタ	R	ID RESET	ID CLK	不定					ID DATA
		W			CS ID	0	0	0	0	0

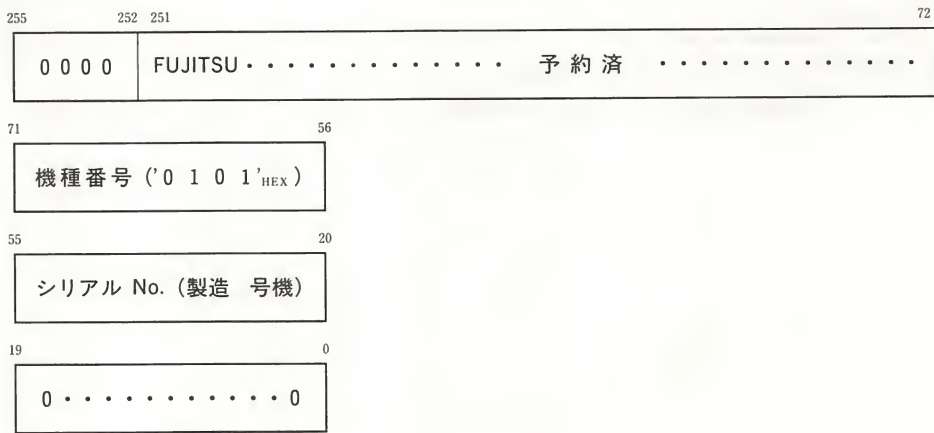
ID RESET (bit7)　：チップセレクトがアクティブで，クロックが1のとき，このビットを0-1-0に変化させるとシリアルROM内のアドレスが0にもどる．

ID CLK (bit6)　：シリアルROM用クロック．チップセレクトがアクティブで，ID RESETが0のときにビットを0から1にすると，シリアルROMのアドレスが1つ進む．

CS ID (bit5)　：シリアルROMのチップセレクト．
0＝アクティブ
1＝インアクティブ

ID DATA (bit0)　：シリアルROM内のアドレスで示されたシリアルデータ．

256ビットのシリアルROMには，以下のフォーマットでデータが書かれている．

データフォーマット

データの内容

アドレス　255〜252　：16進1桁0H固定

アドレス　251〜72　：予約済(将来使用予定)
アドレス　251〜224は，FUJITSU(46H，55H，4AH，49H，54H，53H，55H)が入る．
アドレス　223〜72は，すべてFH．

アドレス　71〜56　：機種番号　16進4桁（FMTOWNSは0101H固定．）

アドレス　55〜20　：シリアルNo．16進9桁（表示文字　0〜9，A〜）

アドレス　19〜0　：16進5桁 00000H 固定（情報が書かれていることを表す．）

●システムステータスレジスタ

システムステータスレジスタ(表I-3-39)は，ディスプレイの解像度の読み出しと，DMA転送先のメモリの種類の設定を行います．

▼表I-3-39　システムステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0400H	システムステータスレジスタ	R	不定							解像度

解像度(bit 0)　：中解像度と高解像度の判断を行う．
0＝固定(中解像度)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0404H	システムステータスレジスタ	R	MAIN MEM	不定						
		W		0	0	0	0	0	0	0

MAIN MEM(bit7)　：メインメモリかVRAMかを選択します．
リセット時，グラフィックVRAM
0＝VRAM
1＝メインメモリ

●メモリ切り換えレジスタ

メモリ切り換えレジスタ(表I-3-40)は，アドレスF8000H～FFFFFHをRAM/ROMのいずれに貼り付けるかの選択と，RAMにするか辞書・学習RAMにするかの選択を行います．

▼表I-3-40　メモリ切り換えレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0480H	メモリ切り換えレジスタ	R	不定						RAM	辞書ROM
		W	0	0	0	0	0	0		

RAM(bit1)　：000F8000H～000FFFFFHをRAM(32KB)または，ROM(32KB)にするかを選択する．
0＝ブートROM
1＝RAM

辞書ROM(bit0)　：RAMにするか辞書・学習RAMにするかを選択する．
0＝RAM
1＝辞書・学習RAM

●辞書レジスタ

辞書レジスタ(表I-3-41)は，辞書ROMのバンクを指定します．

▼表I-3-41　辞書レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0484H	辞書ROM	R	不定				DBK3	DBK2	DBK1	DBK0
		W	0	0	0	0				

32KBで16バンク(512KB)ROMを扱うことができる．

バンク		DBK3	DBK2	DBK1	DBK0
バンク0	辞書ROMバンク0	0	0	0	0
バンク1	辞書ROMバンク1	0	0	0	1
バンク2	辞書ROMバンク2	0	0	1	0
バンク3	辞書ROMバンク3	0	0	1	1
⋮	⋮	⋮	⋮	⋮	⋮
バンク15	辞書ROMバンク15	1	1	1	1

●メモリカードステータス

メモリカードステータス(表I-3-42)は，メモリカード(ROMカードスロット0)の抜き差しを監視するレジスタです．

すなわち，現在のカードの有無はCD-0-1で参照できますが，抜き差しが行われたかどうかについても，CHANGEで調べることができます．CHANGEは，抜き差し動作で1にセットされ，参照のためステータスを読み出した直後に0にクリアされます．

また，バックアップ用のバッテリの残量は，段階別に，

RED　…………バックアップ不能(バックアップされたデータは保証されない)

YELLOW　……残りが少ない(交換を要す)

の各ビットで示されます．いずれかのビットが1になったときは，電池交換を含めて，なんらかの対応が必要になります．例えば，YELLOWの場合，電池交換の際には，バックアップ中のデータはディスクなどにセーブしておかないと消えてしまう点や，交換後にロードして復元を必要とすることなどへのソフトウェア上の対応がそれです．

メモリカードへのライトプロテクトは，WPで調べることができます．このビットが0ならば，書き込みが可能です．

▼表I-3-42　メモリカードステータス

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
048AH	メモリカードステータス	R	CHANGE	不定	RED	YELLOW	(RDY)	CD-0	CD-1	WP

CHANGE (bit7)　：メモリカード(ROMカードなど)の抜き差しが行われたことを示す．
リードすることにより0になる．
1＝抜き差しが行われた

RED，YELLOW (bit5，4)　：バックアップ電池の残量を示す．
REDはバックアップ電池の保証ができないことを示す．
1＝電池の容量がない
YELLOWはバックアップ電池交換を示す．
1＝電池交換が必要

CD-0，1 (bit2，1)　：カードの有無を示す．

CD-1	CD-0	機　能
0	0	カードあり
0	1	不完全挿入
1	0	不完全挿入
1	1	カードなし

WP (bit0)　：メモリカードユニットの書き込み禁止を示す．
0＝書き込み可能
1＝書き込み禁止

(RDY) (bit3)　：未使用．
(EEPROMを搭載するメモリカードユニットでEEPROMが書き込み可能状態であることを示す．
0＝書き込み不可
1＝書き込み可能)

●拡張NMI関係のレジスタ

拡張バスからのNMIに対応するレジスタとしては，NMIマスクレジスタ(表I-3-43)と，NMIステータスレジスタ(表I-3-44)があります．

NMIマスクレジスタは，拡張バスからのNMI要求をマスク(無視)するのに使用されます．このレジスタのBNMIが0のとき，拡張バスからのNMIがマスクされます．NMI(マスクできない割り込み)にマスクをかけるという，この設計仕様の意味するところは，通常はマスクし，必要のあるときのみイネーブル(受け付ける)にするということです．

NMIステータスレジスタは，NMIの対応ハンドラが，拡張バスからのNMIかどうかを調べるためのレジスタです．このレジスタのBNMIは，拡張バスからNMIが発生したときに1になり，拡張していないときには常に0を保ちます．

▼表I-3-43　NMIマスクレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05C0H	NMIマスクレジスタ	R	不　定				BNMI	不　定		
		W	0	0	0	0		0	0	0

BNMI(bit3)　　：拡張バスNMIマスクを示す．
　　0＝マスク
　　1＝イネーブル

▼表I-3-44　NMIステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05C2H	NMIステータスレジスタ	R	不　定				BNMI	不　定		

BNMI(bit3)　　：拡張バスNMIを示す．
　　0＝なし
　　1＝あり

●TVRAM書き込みレジスタ

TVRAM書き込みレジスタ(表I-3-45)は，テキストVRAM領域への書き込みを行ったかどうかをMDで示します．

▼表I-3-45　TVRAM書き込みレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05C8H	TVRAM書き込みレジスタ	R	MD	不　定						

MD(bit7)　　：テキストVRAMの状態を示す．
　リードすることにより0になる．
　　0＝テキストVRAMの書き込みを行わなかった
　　1＝テキストVRAMの書き込みを行った

●VSYNC割り込み原因クリアレジスタ

VSYNC割り込み原因クリアレジスタ(表I-3-46)は，VSYNC割り込みが発生したとき，割り込み対応ハンドラでVSYNC割り込み要求を停止させるためのダミーレジスタです．

特にビットごとの意味はありませんが，このレジスタに書き込みを行うことで要求解除が行えます．

▼表 I-3-46　VSYNC割り込み原因クリアレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05CAH	VSYNC 割り込み原因クリアレジスタ	W	WRITE → クリア							

ダミーライトすることにより，VSYNC割り込み原因をクリアする．

● FIRQ レジスタ

FIRQ レジスタ(表 I-3-47)のビット 7 は，ライトペン割り込み要求フラグとしての役割がありますが，FMTOWNS ではライトペンをサポートしていないので，常に 0 です．

▼表 I-3-47　FIRQレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF84H	FIRQレジスタ	R	0	不　定						

bit7　：ライトペン割り込み要求フラグ．
FMTOWNSでは，サポートしていないので常に0．

●漢字 CG アクセスレジスタ

漢字 CG アクセスレジスタ(表 I-3-48)は，漢字 CG(キャラクタジェネレータ)の漢字フォントをリード／ライトするために使用します．

▼表 I-3-48　漢字CGアクセスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF94H	漢字CG アクセスレジスタ	R	L2CG	不　定						
		W	KC15	KC14	KC13	KC12	KC11	KC10	KC9	KC8
000C FF95H		W	KC7	KC6	KC5	KC4	KC3	KC2	KC1	KC0
000C FF96H		R/W	D15	D14	D13	D12	D11	D10	D9	D8
000C FF97H		R/W	D7	D6	D5	D4	D3	D2	D1	D0

L2CG(bit7)　：第2水準漢字CGの有無を示す．このビットは常に1(第2水準漢字CG有)を示す．

KC15-0　：アクセスする漢字のコードをJISコードで指定する．

D15-0　：漢字CGのリード/ライトデータ．フォントデータはROWスキャンでリード/ライトでき，FF97番地のアクセスによってROWアドレスがインクリメントされる．
FF95番地のアクセスによってROWアドレスがクリアされる．
読み出すごとにROMの下位4ビットはカウントされる．

●ブザー制御レジスタ

ブザー制御レジスタ(表Ⅰ-3-49)は，ブザーをON／OFFするために使用します．

▼表Ⅰ-3-49 ブザー制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF98H	ブザー制御レジスタ	R/W	READ→ON WRITE→OFF							

このレジスタをリードするとONになり，ライトするとOFFになる．

●漢字VRAMレジスタ

漢字VRAMレジスタ(表Ⅰ-3-50)は，漢字VRAMとANKCGのどちらをアクセスするかを決めるものです．

▼表Ⅰ-3-50 漢字VRAMレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF99H	漢字VRAMレジスタ	W	0	0	0	0	0	0	0	ANKCG

ANKCG(bit0)　：漢字VRAMとANKCGのどちらをアクセスするかを指定する．
　0＝漢字VRAMを選択する
　1＝ANKCGを選択する

●論理演算レジスタ

論理演算レジスタ(表Ⅰ-3-51)のビットは，常に0に固定です．

▼表Ⅰ-3-51 論理演算レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FFA0H	論理演算レジスタ	R	ESTART (0)	不　定						

ESTART(bit7)　：論理演算は，常に0固定である．

第4章

表示システム

この章では，FMTOWNSの複雑な画面表示が，ハードウェアでどのように行われているか，について解説します．

具体的には，画面の種類，VRAMの読み書き，パレット，スプライト，ディスプレイへの出力，スクロールなどの仕組みについて取り上げます．

4.1 画面表示の概要

FMTOWNSには優れた画面表示機能があり，解像度，同時表示色数の異なるさまざまな画面表示が可能です．基本的な画面表示のモードとしては，18種類の画面モードがあります．そしてスクロール，デジタイズ，スーパーインポーズ，スプライトなどが使用できるかどうかは，画面モードによります．この節では，画面表示の概要を解説します．表示機能の詳細については，次節以降を参照してください．

4.1.1 画面モードと表示機能

表I-4-1は，画面表示機能を画面モードごとに整理したものです．

●画面モード

基本的な画面モードが18種類あります．

任意の画面モードにするには，各種のレジスタの設定が必要です．詳しくは「4.7.3 CRTCのレジスタとその設定例」を参照してください．

▼表I-4-1　画面モード一覧

画面モード番号	仮想画面	表示画面(有効ピクセルサイズ)	実際の表示領域	同時表示色	パレット	画面数	スクロール	CRT水平周波数	スプライト	スーパーインポーズ	ビデオデジタイズ
1	640×400	640×400	640×400 ノンインタレース (FMR-50互換)	16色	16/ 4096色	2面	なし	24.37KHz アンダースキャン	使用不可	使用不可	使用不可
2		640×200	640×200 ノンインタレース (2度読み)								
3	1024×512	640×480 (縦横比1:1)	640×480 ノンインタレース				円筒 (横方向制限付)	31.47KHz アンダースキャン			
4		640×400	640×400 ノンインタレース					24.37KHz アンダースキャン			
5	256×512	256×256	256×256 ノンインタレース	32768色	なし			31.47KHz アンダースキャン	使用可		
6		256×256	256×256 ノンインタレース					24.37KHz アンダースキャン			
7		256×240	230×216 インタレース					15.73KHz オーバースキャン		使用可	使用可
8		256×240	230×216 インタレース								
9	512×256	360×240	324×216 インタレース				球面 (無制限)		使用不可		
10		320×240	320×240 インタレース					31.47KHz アンダースキャン		使用不可	使用不可
11		320×240	288×216 ノンインタレース					15.73KHz オーバースキャン		使用可	使用可
12	1024×512	640×480 (縦横比1:1)	640×480 ノンインタレース	256色	256/ 1677万色	1面	円筒 (横方向制限付)	31.47KHz アンダースキャン		使用不可	使用不可
13		640×400	640×400 ノンインタレース					24.37KHz アンダースキャン			
14		720×480	648×432 インタレース					15.73KHz オーバースキャン		使用可	
15	512×512	320×480 (縦横比2:1)	320×480 ノンインタレース	32768色	なし			31.47KHz アンダースキャン		使用不可	
16		320×480 (縦横比2:1)	288×432 インタレース					15.73KHz オーバースキャン		使用可	使用可
17		512×480	512×480 ノンインタレース					31.47KHz アンダースキャン		使用不可	使用不可
18		512×480	512×432 インタレース					15.73KHz オーバースキャン		使用可	使用可

●仮想画面と表示画面

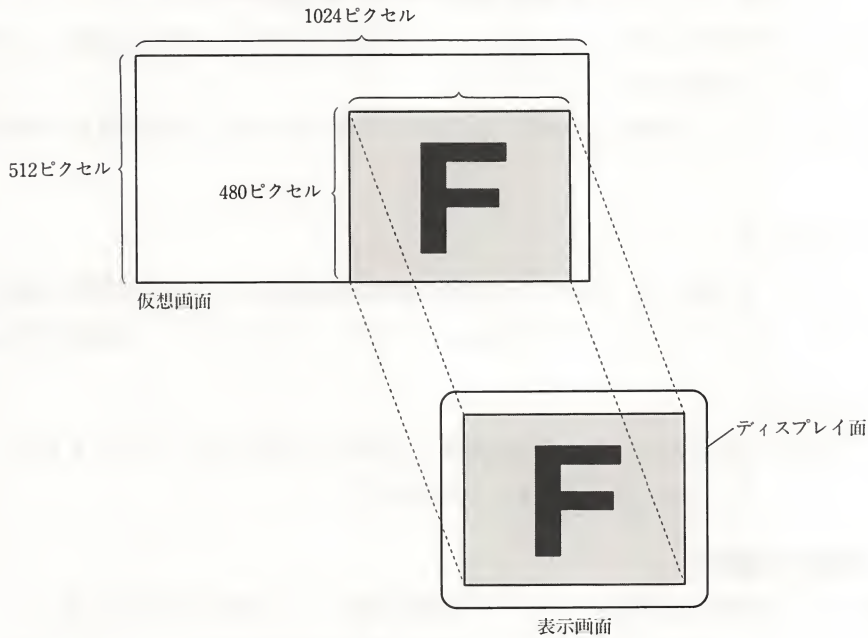
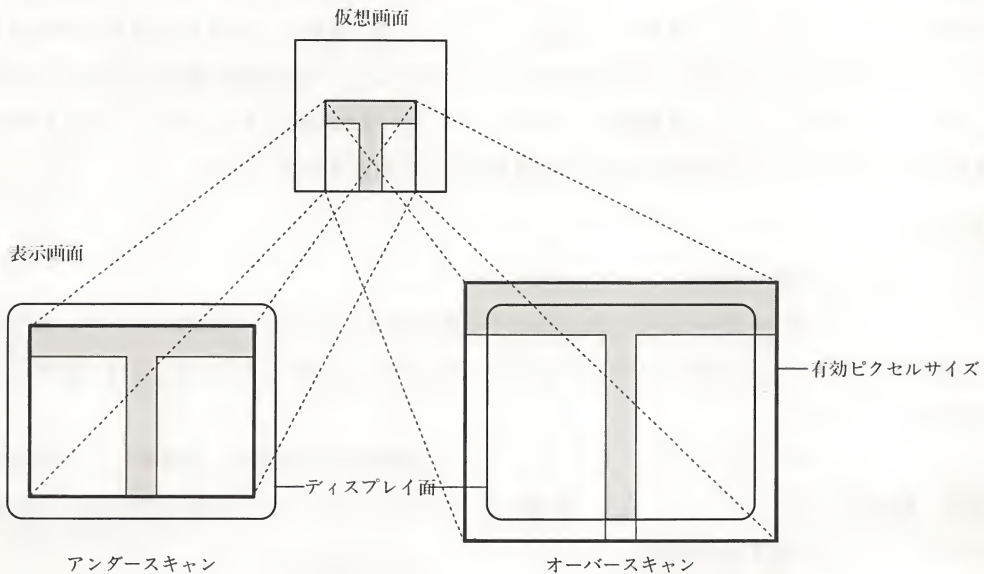
FMTOWNSではビットマップによる画面表示を採用しています．ビットマップは，画像データを画素(ピクセル)単位でメモリ(VRAM)に格納するものです．文字の表示もこの方法で行っています．

そして，画面の概念として，仮想画面と表示画面の2段階がハードウェアでサポートされています．VRAMと直接対応しているのが仮想画面で，この仮想画面の範囲からディスプレイ表示のために切り出した部分を，表示画面といいます．仮想画面と表示画面の関係は図I-4-1のようになります．

ディスプレイの表示には，アンダースキャンとオーバースキャンの2とおりがあります．アンダースキャンの場合は，表示画面のすべての範囲がディスプレイに表示されますが，オーバ

ースキャンの場合は，表示画面のうちの周辺部は実際には表示されません(図Ｉ-4-2)．表示画面の周辺部を含めた範囲を有効ピクセルサイズといいます．

表示画面に表示されている仮想画面の範囲を変更すると，画面がスクロールします．FM TOWNS では円筒スクロールの外に，球面スクロール(後述)が可能です．また，２枚の仮想画面を重ねて表示させることもできます．

▼図Ｉ-4-1　表示画面と仮想画面の関係(画面モード３の場合)

▼図Ｉ-4-2　アンダースキャンとオーバースキャン

●同時表示色とパレット

画面に表示可能な色数は，表示する内容に合わせて選択できます．

例えば，32768色のモードは，自然画の表示に適しています．また，パレット機能を使うと，16色モードでは4096色中16色，256色モードでは1677万色中256色を選択できるので，微妙な色の差を表現することができます．

●スプライト

スプライトは，画面上でパターンを高速に動かすために使われます．

通常の画面の手前に重ねて表示され，表示アドレスを変えるだけで，背後の画面データを書き変えることなしに，移動させることができます．

最大1024個のスプライトが表示できます．スプライトの各ピクセルには32768色までの色が付けられます．

●スーパーインポーズ

スーパーインポーズ機能とは，外部ビデオ信号の画面の手前にコンピュータ画面を重ねて表示するものです．スーパーインポーズを行うには，オプションのビデオカードが必要です．

●ビデオデジタイズ

外部ビデオ信号をデジタルデータに即時に変換し，VRAMに取り込むことができます．デジタイズを行うには，オプションのビデオカードが必要です．

●FMR-50互換の画面表示

FMTOWNSは，FMR-50用のソフトウェアが使用可能なように設計がされています．

画面表示についても，FMR-50とみかけ上，同等の表示を行うためのハードウェア上の工夫がされています．

FMR-50には，フレームアクセスのできるグラフィックVRAMと文字を表示するためのテキストVRAMがありますが，FMTOWNSでは，FMR-50と互換の画面表示を実現するために，フレームアクセスのできる画面モードと，テキストVRAMをエミュレートするためのRAM8KB(テキストVRAM4KB，漢字VRAM4KB)が用意されています．

●その他

表I-4-1中のその他の用語について説明します．

インタレースとは，画面を表示する際，画面を奇数ラスタ(ラスタとは輝線のことで，水平方向の線1本に相当する)と偶数ラスタに分けて交互に表示する方式です．この方式は一般のテレビで使われています．

インタレースについて詳しくは，「4.7.2 ブラウン管の表示の仕組み」を参照してください．

なお，縦横比とは，ディスプレイ上で縦横同数のピクセルによって正方形を描いたときの縦横の大きさの比率を表すものです．

4.2 画面制御系のハードウェア概要

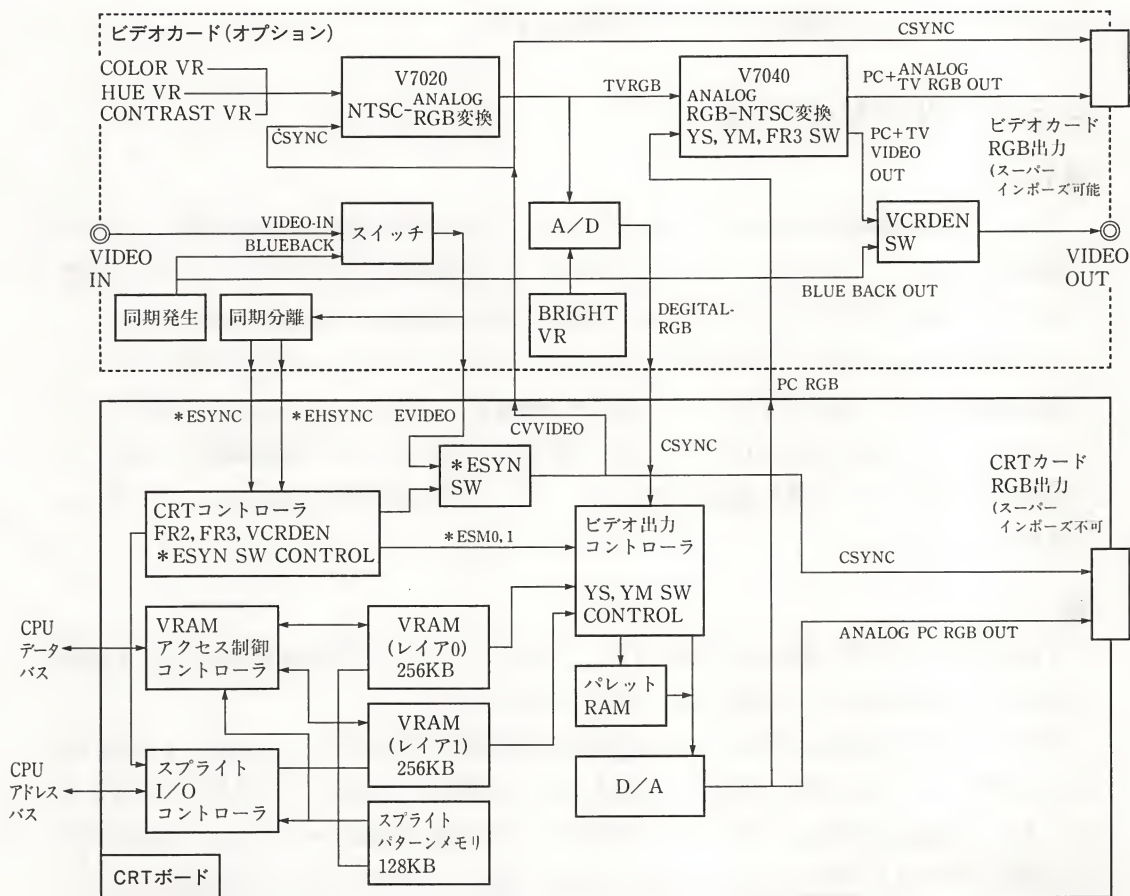
この節では，画面制御において重要な役割を果たしているCRT制御部の概要について説明します．

4.2.1 CRT制御部

画面制御系ブロック図を，図I-4-3に示します．

画面制御の中心的役割を担っているのがCRT制御部です．CRT制御部の最も大きな働きは，VRAMに書かれたデータをディスプレイに出力することです．

CRT制御部の出力は，ディスプレイに直接接続できるアナログRGB形式です．この出力は，コンピュータ画像のみを出力するもので，ビデオカードのアナログRGB出力と違ってスーパーインポーズした画像の出力はできません．

▼図I-4-3　画面制御系ブロック図

CRT制御部には，4個のCRT関係のコントローラがあります．

VRAMアクセス制御コントローラは，VRAMの読み書きを制御するものです．CRTコントローラは，ディスプレイ表示のタイミングを制御します．また，スプライトI/Oコントローラはスプライト画面を制御し，ビデオ出力制御コントローラは，表示するビデオ信号の合成などの制御を行っています．

各コントローラには，VRAM，スプライトパターンメモリ，パレットRAM，D/Aコンバータなどがつながっています．

それぞれのLSIを制御するために種々のレジスタが用意されており，CPUからI/Oアクセスによって機能します．個々のレジスタの仕組みと働きについては，次節以降を参照してください．ビデオカードについては，「4.11　ビデオカード」を参照してください．

4.3 VRAM

この節では，VRAMがどのような方法で読み出されているかについて述べるとともに，VRAMのアドレスとの関係などについても説明します．

4.3.1 VRAMとページ

●VRAM

ディスプレイに静止画を表示するには，コンピュータから同じ画像信号を繰り返し送り続ける必要があります．このため，コンピュータには，この画像信号をデジタルデータの形で記憶するメモリが用意されています．これが，VRAM(Video Random Access Memory)です．

FMTOWNSには，512KBのVRAMがあります．このメモリはメインメモリと同じアドレス空間に配置されているので，CPUから自在に読み書きができます．また，アクセスの窓口として2組のアドレス線が用意されており，同じVRAMをCPUとビデオ制御回路から並行して読み書きできるので，高速な描画が可能です．このようなVRAMの形式をデュアルポートVRAMといいます．

●ページ

VRAMは，全体を1画面の仮想画面として使うことや，2画面(256KBずつ)，4画面(128KB)の仮想画面に分けて使用することができます．

例えば，仮想画面が1024×512ピクセルで同時表示色数256色の画面モードでは，1画面の表示に必要なメモリは512KBであり，VRAM全体を仮想画面1画面として使うことになります．また，仮想画面が1024×512ピクセルで同時表示色数16色の画面モードでは，1画面の表示に必要なメモリは256KBであり，2画面の使用が可能です．(図I-4-3)．

VRAMの１画面をページといいます．２分割の場合，VRAMのアドレスの若い方から，ページ０～１になります．４分割の場合は，ページ０～３となります．

仮想画面の１画面当たりに必要なVRAMの大きさは，総ピクセル数×１ピクセル当たりのVRAMのビット数という式で求められますが，１ピクセル当たりのVRAMのビット数は同時表示色の数によって表I-4-2のようになります．

▼表I-4-2　同時表示色数と１ピクセル当たりのVRAMのビット数

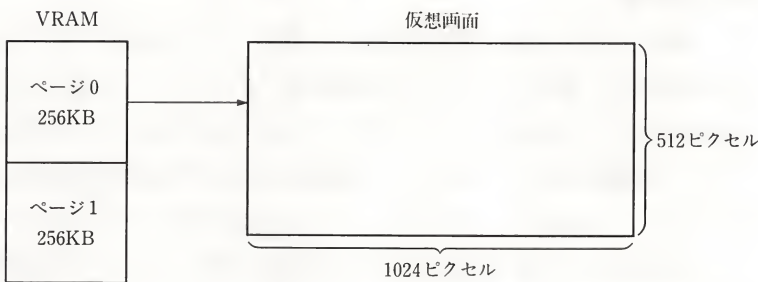
同時表示色数	１ピクセル当たりのVRAMのビット数
16色	4ビット(2^4)
256色	8ビット(2^8)
32768色	16ビット(2^{16})

なお，32768色時には16ビット中の１ビットをスーパーインポーズ制御に使うため，色数を決めるビット数は15ビットです．

4.3.2　画面レイアと画面の重ね合わせ

FMTOWNSでは，VRAMの値をディスプレイに表示する際に，画面レイアという概念を用います．画面レイアは，いわば，〝画面表示のためVRAMの層〟です．

画面レイアは２個用意されており，それぞれを画面レイア０，画面レイア１と呼びます．これらの画面レイアにVRAMの各ページを割り当て，画面表示を行います（図I-4-4）．２個の画面レイアは重ねて表示することができ，優先度の高い方が手前に表示されます．

▼図I-4-4　VRAMと仮想画面の関係（画面モード３，４の場合）

●画面レイアとページの関係

VRAMのページと画面レイアの関係は次のようになります．

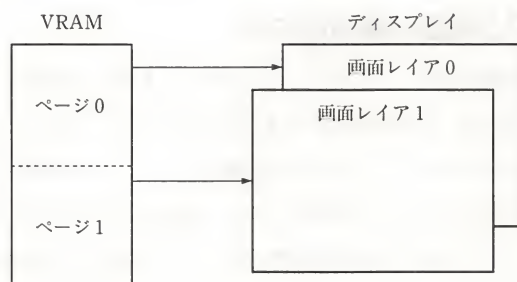
VRAM全体で画面１枚を表示する場合には，ページ０が画面レイア０に必ず割り当てられます．VRAMを２ページに分けて使用した場合には，ページ０が画面レイア０に，ページ１が画面レイア１に必ず割り当てられます（図I-4-5）．

VRAMを4ページに分けて使用した場合(640×400ピクセルの画面)は，画面レイア0にページ0またはページ1が，画面レイア1にページ2またはページ3が割り当てられます．ただし，実際に表示されるのは，ページ0とページ1のどちらかと，ページ2とページ3のどちらかとなります．

さまざまな画面のステータスの設定は，画面レイアごとに可能ですから，異なった画面モードの画面を重ねて表示することができます．

4分割の場合,画面レイア0にページ0とページ1のどれを割り当てるか(どちらを表示するか)は後述のグラフィックVRAMディスプレイモードレジスタ(表I-4-41)によって決まります．また，アクセスページの切り替えはグラフィックVRAMページセレクトレジスタ(表I-4-44)によって行います．

画面レイア1は，CRTCのフレーム先頭アドレス1をページ2または，3の先頭アドレスに設定することによって，表示ページが決まります．

▼図I-4-5　画面レイアとページの関係

●画面の重ね合わせ順位

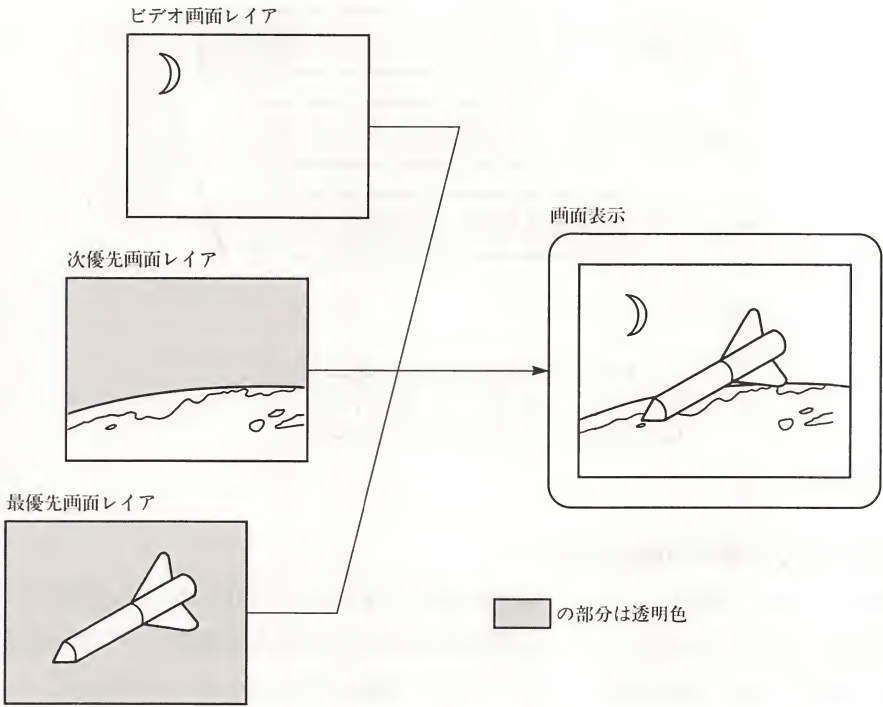
前述のように，同時に2枚の画面を重ねて表示した場合，2つの画面レイアのどちらを前面に表示するか(優先順位)は,後述するビデオ出力制御のプライオリティレジスタ(表I-4-37)に設定します．PRIが0のときは画面レイア0が前，1のときは画面レイア1が前となります．

スーパーインポーズ機能を使ってコンピュータ画面と外部ビデオ信号を合成することもできます．この場合には，2つの画面レイアの外に，ビデオ信号の表示のためのビデオ画面レイアを使います．ビデオ画面レイアは，画面レイアより優先度が低いため，ビデオ画像は常にコンピュータの画像の背後に表示されます．

例えば，ロケットを描いたページを最優先画面レイアに，地球を描いたページを残りの画面レイアに，ビデオ画面レイアに宇宙と月の画像を割り当てると，ディスプレイには，宇宙空間で地球のまわりをロケットが飛んでいるようすが表示されます(図I-4-6)．

なお，表示されているコンピュータ画面が1画面の場合でも，スーパーインポーズは可能です．

スーパーインポーズについては，「4.11　ビデオカード」を参照してください．

▼図Ⅰ-4-6　複数画面レイアの合成例

●画面重ね合わせ時の透過処理

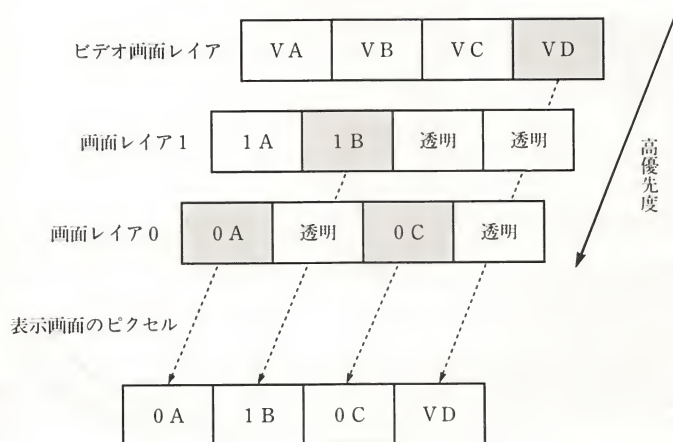
画面の色には透明色が設定できます．優先度の高い画面レイアの色が透明の場合には同じ座標上の優先度の低い画面レイアのドットの色が見えることになります．

透明色とピクセルデータとの関係は，同時表示色数によって異なり，表Ⅰ-4-3のようになります．

ビデオ画面も含めて図示したのが，図Ⅰ-4-7です．透過の処理により，透明を除いて最も優先度の高い画面レイアのピクセルの色が表示されることになります．

▼表Ⅰ-4-3　同時表示色数と透明色を表すピクセルデータの関係

同時表示色数	透明色を表すピクセルデータ
32768色(16ビット)	最上位ビット(ビット15)が1
256色(8ビット)	ビット構成が　00000000
16色(4ビット)	ビット構成が　0000

▼図Ⅰ-4-7　透過処理の概念(画面レイア0優先の場合)

●異なったモードの画面の重ね合わせ

画面のモードは，画面レイアごとに個別に設定できるので，画面レイア1を画面モード5(256×256ピクセル－32768色)にして，風景写真を表示し，その上に画面レイア0を画面モード3(640×480ピクセル－16色)にして，文字を重ねて表示するということが可能です．

どの画面の重ね合わせが可能かどうかは，「4.7.3　CRTCのレジスタとその設定例」を参照してください．

4.3.3 VRAMの読み書き

VRAMの内容は，CPUから直接，読み書きすることができます．

ここでは，その仕組みについて解説します．

●CPUのレジスタ～VRAM間のデータ転送

FMTOWNSでは，VRAMのデータの並び順が32ビット転送命令に適した順になっており，1ピクセル当たりのビット数にかかわらず，画面のピクセルの並び順と逆の順でピクセル単位にレジスタに値を格納し，32ビット転送命令を実行すればいいようになっています．

この方法の採用により，ピクセルデータの高速な転送が可能です．32ビットにまとめられた複数のピクセルのことを，パックドピクセルと呼びます．

CPUのレジスタとVRAMおよび，画面の対応関係は，表示色数(ピクセル当たりのビット数)によって，図Ⅰ-4-8，図Ⅰ-4-9，図Ⅰ-4-10のようになっています．

すなわち，レジスタとメモリ間の転送によって生ずるビットの並びの違いは表示の際に補正されます．

▼図 I-4-8　256色データ転送時のレジスタ，VRAM，画面表示

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

レジスタ

ピクセル 3								ピクセル 2								ピクセル 1								ピクセル 0							
C7	C6	C5	C4	C3	C2	C1	C0	C7	C6	C5	C4	C3	C2	C1	C0	C7	C6	C5	C4	C3	C2	C1	C0	C7	C6	C5	C4	C3	C2	C1	C0

転送命令

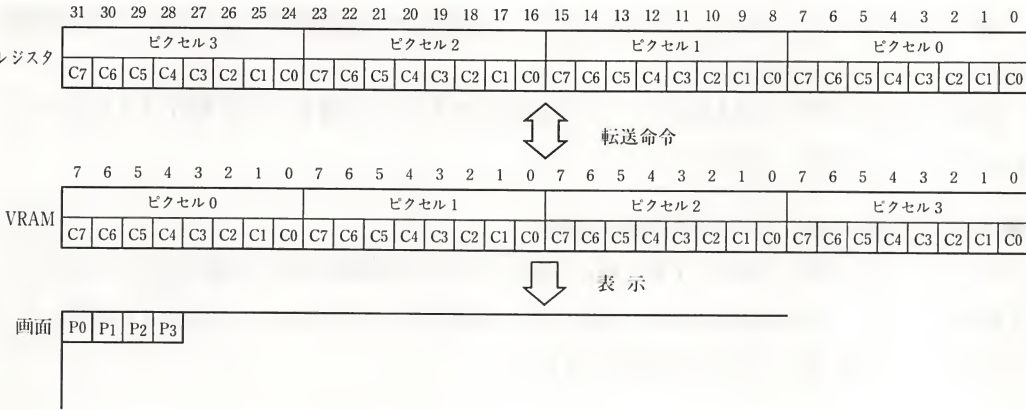
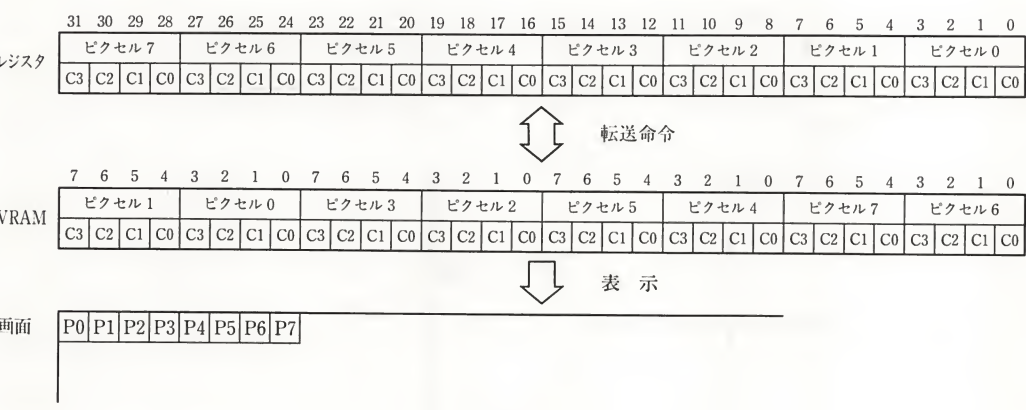
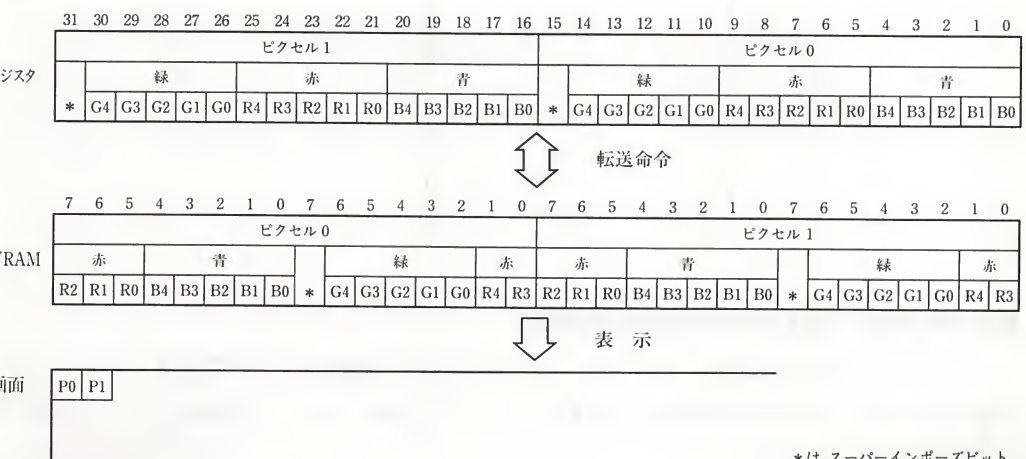
7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0 7 6 5 4 3 2 1 0

VRAM

ピクセル 0								ピクセル 1								ピクセル 2								ピクセル 3							
C7	C6	C5	C4	C3	C2	C1	C0	C7	C6	C5	C4	C3	C2	C1	C0	C7	C6	C5	C4	C3	C2	C1	C0	C7	C6	C5	C4	C3	C2	C1	C0

表示

画面

P0	P1	P2	P3

▼図 I-4-9　16色データ転送時のレジスタ，VRAM，画面表示

▼図 I-4-10　32768色データ転送時のレジスタ，VRAM，画面表示

4.3.4 VRAMのアドレスマップ

FMTOWNSでは，512KBのVRAMに対して，複数のメモリ配置がされており，画面の種類によって，アドレスが異なります．

各仮想画面におけるVRAMのアドレス配置を図I-4-11，図I-4-12，図I-4-13，図I-4-14，図I-4-15，図I-4-16に示します．

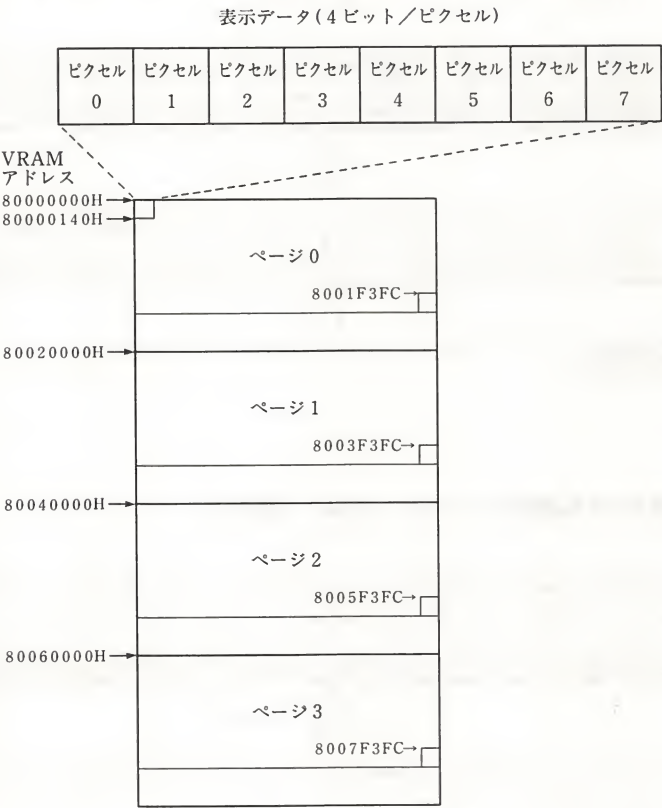
●仮想画面640×400ピクセル16色表示の場合

4ビット／ピクセルのため，CPUから一度にアクセス可能なピクセル数は8ピクセルです．VRAMのアドレスは80000000Hから始まります．画面レイア0にはページ0か1，画面レイア1にはページ2か3のいずれかを選択できます．

▼図I-4-11　仮想画面640×400ピクセル(16色表示)のVRAMアドレスマップ

表示データ(4ビット／ピクセル)

ピクセル0	ピクセル1	ピクセル2	ピクセル3	ピクセル4	ピクセル5	ピクセル6	ピクセル7

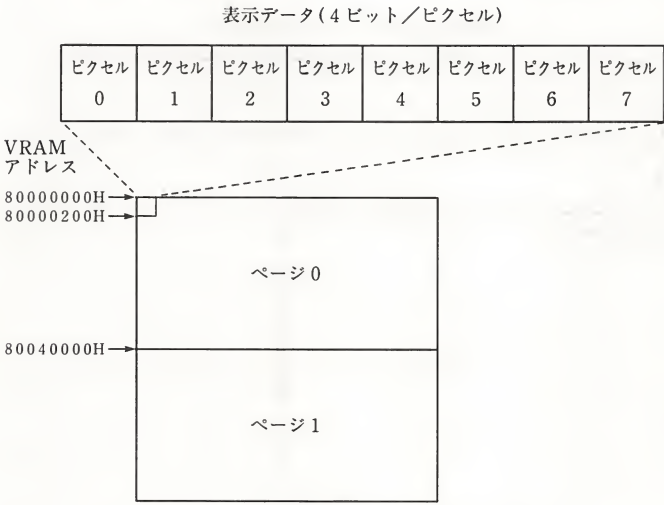
VRAMアドレス
80000000H→
80000140H→
ページ0
8001F3FC→
80020000H→
ページ1
8003F3FC→
80040000H→
ページ2
8005F3FC→
80060000H→
ページ3
8007F3FC→

●仮想画面1024×512ピクセル16色表示の場合

4ビット／ピクセルのため，CPUから一度にアクセス可能なピクセル数は8ピクセルです．VRAMのアドレスは80000000Hから始まります．ページ0，ページ1が画面レイア0，画面レイア1に対応しています．

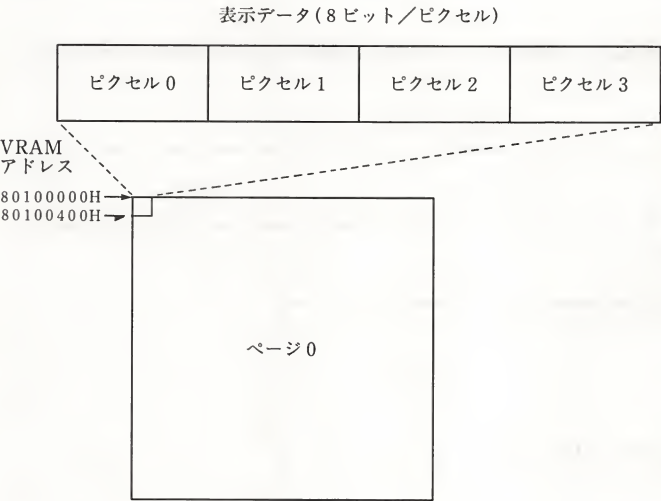
▼図 I-4-12　仮想画面1024×512ピクセル(16色表示)の VRAM アドレスマップ

表示データ(４ビット／ピクセル)

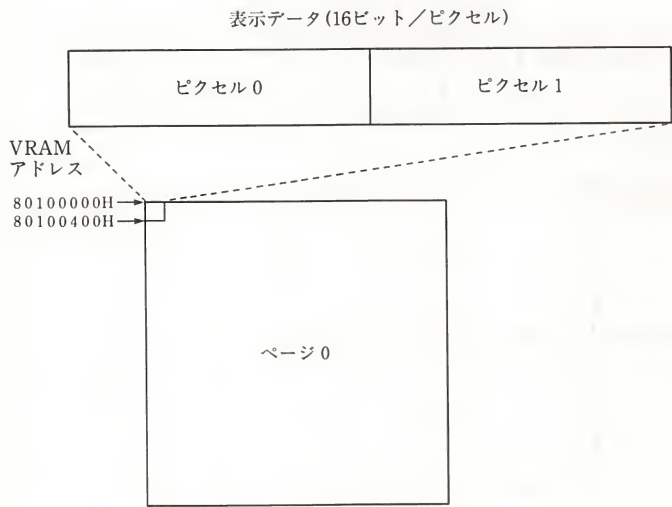
ピクセル 0	ピクセル 1	ピクセル 2	ピクセル 3	ピクセル 4	ピクセル 5	ピクセル 6	ピクセル 7

VRAM
アドレス
80000000H
80000200H
ページ０
80040000H
ページ１

●仮想画面1024×512ピクセル256色表示の場合

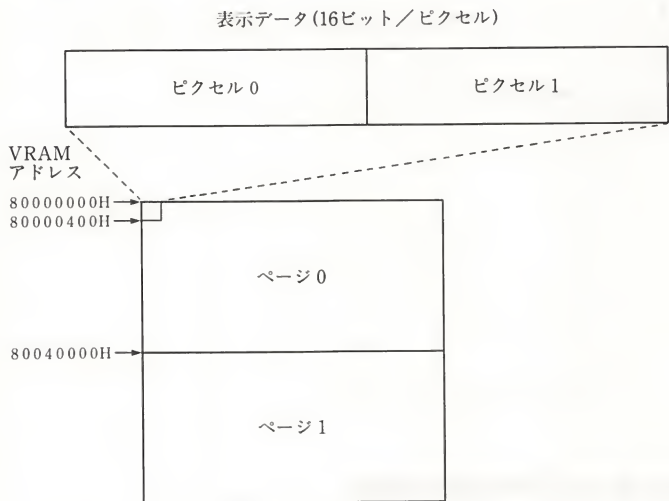
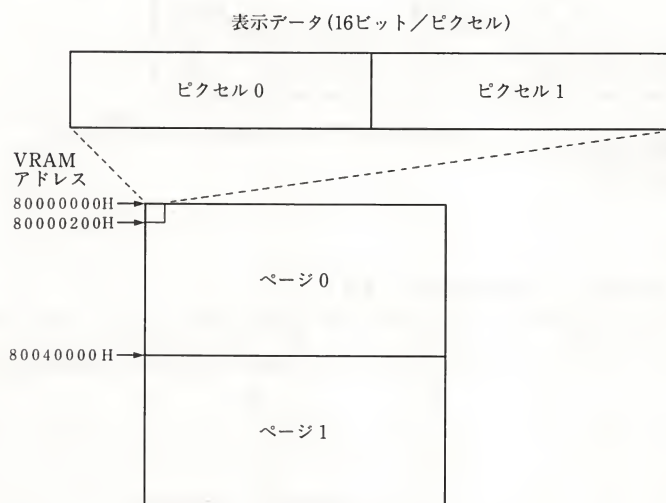
８ビット／ピクセルのため，CPU から一度にアクセス可能なピクセル数は４ピクセルです．VRAM のアドレスは80100000Hから始まります．１画面だけしか取れません．

▼図 I-4-13　仮想画面1024×512ピクセル(256色表示)の VRAM アドレスマップ

●仮想画面512×512ピクセル32768色表示の場合

16ビット／ピクセルのため，CPU から一度にアクセス可能なピクセル数は２ピクセルです．VRAM のアドレスは80100000Hから始まります．１画面だけしか取れません．

▼図Ⅰ-4-14　仮想画面512×512ピクセル(32768色表示)のVRAMアドレスマップ

●仮想画面512×256ピクセル32768色表示と仮想画面256×512ピクセル32768色表示の場合

16ビット／ピクセルのため，CPUから一度にアクセス可能なピクセル数は2ピクセルです．VRAMのアドレスは80000000Hから始まります．ページ0，ページ1が画面レイア0，画面レイア1に対応しています．

▼図Ⅰ-4-15　仮想画面512×256ピクセル(32768色表示)のVRAMアドレスマップ

▼図Ｉ-4-16　仮想画面256×512ピクセル(32768色表示)のVRAMアドレスマップ

4.3.5　VRAMアクセス制御のレジスタ

VRAMの読み書きの制御に関連するレジスタには次のようなものがあります．

パックドピクセルマスクレジスタ
MIXレジスタ
グラフィックVRAM更新モードレジスタ
グラフィックVRAMページセレクトレジスタ

このうち，後の3つはFMR-50互換モードに関連するレジスタなので，「4.9　FMR-50互換の画面表示機能」で説明し，ここでは，パックドピクセルマスクレジスタのみを説明します．

●パックドピクセルマスクレジスタ0，1

パックドピクセルマスクレジスタ(表Ｉ-4-4)は，CPUからVRAMに書き込みをする際に，ビット単位にマスクをするためのレジスタです．このレジスタの各ビットは，VRAMのピクセルのビット構成に対応させて該当ビットごとにマスク(書き込み阻止)の有無を指定します．マスクするビットは0，書き込むビットは1を設定します．このレジスタへの書き込みは，VRAMアクセスコントローラI/Oレジスタ(表Ｉ-4-5)の0458H番地にレジスタ番号を，045AH番地と045BH番地にデータの下位と上位を設定することによって行います．

▼表I-4-4　VRAMアクセスコントローラの内部レジスタ

レジスタ番号	レ　ジ　ス　タ　名	サイズ
0 0	パックドピクセルマスクレジスタ0（下位ワード）	W
0 1	パックドピクセルマスクレジスタ1（上位ワード）	W

8/16/32ビットでのパックドピクセルアクセス時に指定したビットを書き込みの対象にする.
　0＝マスクする(書き込まない)
　1＝マスクしない(書き込む)

▼表I-4-5　VRAMアクセスコントローラI/Oレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0458H	アドレスレジスタ	R	不定						RA1	RA0
		W	0	0	0	0	0	0		
045AH	データレジスタ(下位)	R/W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
045BH	データレジスタ(上位)	R/W	RD15	RD14	RD13	RD12	RD11	RD10	RD9	RD8

4.4　スクロール

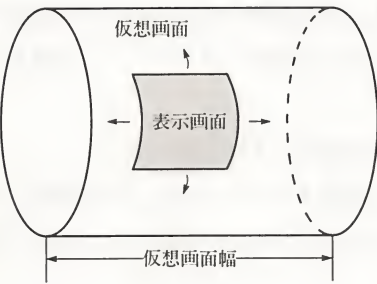
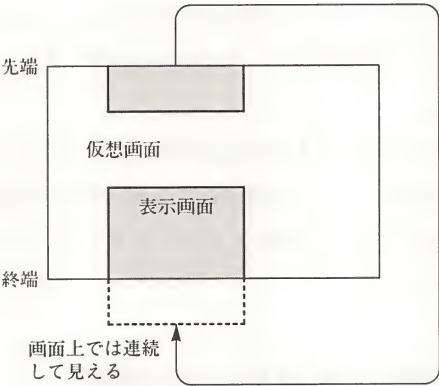
FMTOWNSでは，仮想画面中で表示範囲をずらすことにより，画面をスクロールさせることができます．円筒スクロールの外に，球面スクロールも可能です．この節では，スクロールの概念について説明します．なお，スクロールの設定は，表示アドレス設定関係のレジスタで行います．詳しくは，「4.7.4　CRTCの内部レジスタ」を参照してください．

4.4.1　円筒スクロール

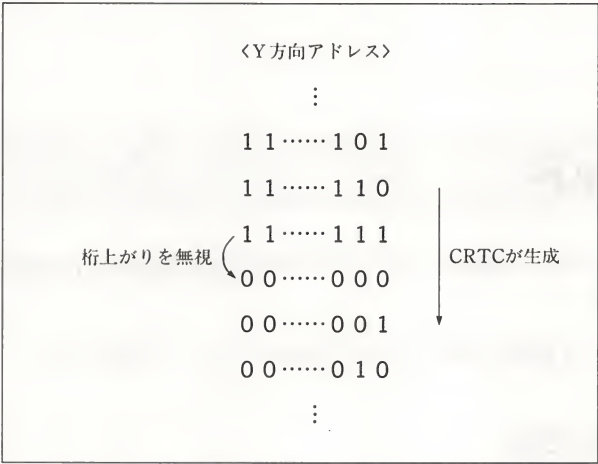
一般にパソコンのテキスト画面では，文字が画面いっぱいに表示されると，次の行を表示する前に画面全体が上に1行押し上げられ，画面のいちばん上の行は画面から見えなくなります．これを垂直スクロールといいます．

この場合，スクロールさせて見ている画面のメモリの先端と終端はつながっていませんが，先端と終端を論理的につなげると，画面上ではあたかもデータが連続しているように見えます．これを円筒スクロールといいます(図I-4-17，図I-4-18)．

FMTOWNSでは，仮想画面の中でディスプレイに表示されている部分(表示画面)を円筒スクロールさせることができます(640×400ピクセルの仮想画面では不可能)．

▼図 I-4-17　円筒スクロールの概念

▼図 I-4-18　円筒スクロールの実際

画面表示は，CRTC(ディスプレイの表示を行っているコントローラ)が，VRAM の内容を読み取ることにより行われています．縦スクロールは，Y軸(垂直)方向のアドレスをカウントアップして実現しているのですが，アドレスの値が最下段(…111B)になったときに，1を加算したときの桁あがりを無視すればY軸方向の VRAM のアドレスは0となり，VRAM の先頭を指します．これが，円筒スクロールの原理です(図 I-4-19)．

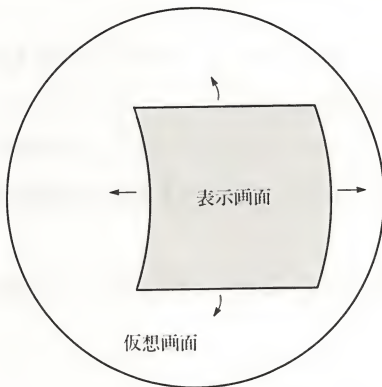
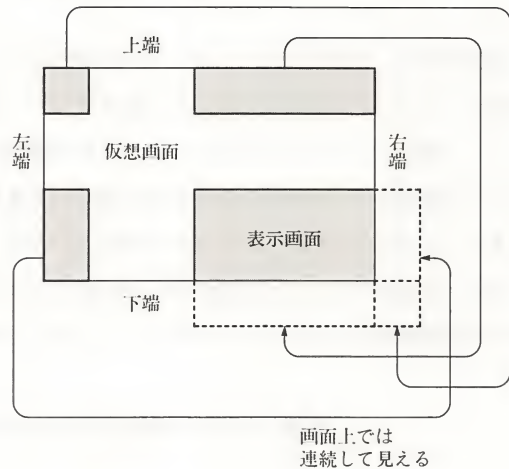
なお，FMTOWNS の仮想画面はX軸(水平)方向にも広がりがあるので，円筒スクロール時にもその範囲内で水平スクロールを行うことができます．

▼図 I-4-19　不連続なアドレスが連続して扱える原理

4.4.2 球面スクロール

円筒スクロールでは，水平方向のスクロールに制約がありましたが，水平方向にも両端のアドレスを連続させ，全方向の連続スクロールを可能にしたのが球面スクロールです(図I-4-20).

512×256ピクセルの仮想画面のときに，球面スクロールを行うことが可能です.

球面スクロール時には，X軸(水平)方向のアドレスも，円筒スクロールのところで説明したような方法で，カウントアップすることによって，連続したスクロールを可能にしています(図I-4-21).

▼図I-4-20 球面スクロールの概念

▼図I-4-21 球面スクロールの実際

4.5 パレット

FMTOWNSでは，パレット機能により，1677万色中256色の表示と，4096色中16色の表示を行うことができます.

この節では，パレット機能を実現している仕組みについて説明します.

4.5.1 色の表示方法

FMTOWNSでは，色の表示に際してパレット機能を使う場合と使わない場合があります.

パレットを使わない場合は，32768色を同時に表示することができます．この場合，仮想画面の1ピクセルのデータはVRAMの16ビットのデータと1対1で対応しており，16ビットのVRAMのデータそのものが色を表しています.

パレットを使う場合は，1677万色(RGB 各８ビット)中256色表示と，4096色中16色表示のどちらかが可能です．

例えば，1677万色中256色表示の場合には，色を記憶できるメモリ領域が256個分用意されています．これをアナログパレットテーブルといいます．各パレットは24ビットの容量があり，RGB 各８ビット(1677万色)のデータを格納できます．

画面表示の際に指定するのは色そのものではなく，このパレットの番号(０～255)です．この番号をパレットコードといいます．したがって，パレットを使う場合に VRAM に置かれているデータは色そのものの番号ではなく，パレットコードということになります．

4.5.2　パレットテーブル

パレットテーブルとは，パレットの色データを格納する領域です．FMTOWNS のパレットテーブルには，FMTOWNS 本来のビットマップ制御用のアナログパレットと FMR-50 互換用に使用されるデジタルパレットがあります．ここではアナログパレットの使い方について説明します．デジタルパレットレジスタについては，「4.9.3　FMR-50 互換のパレットの指定」で解説します．

4.5.3　アナログパレットレジスタ

アナログパレットには，1677万色から選択した256色を格納する256色パレットと4096色から選択した16色を格納する16色パレットの２種類があります．

アナログパレットの読み書きは，アナログパレットレジスタ(表 I-4-6)を通して行います．

このレジスタは，256色パレットと16色パレットの両方に対応しています．アナログパレットに対して色のデータを読み書きする際には，アナログパレットレジスタに対して，アナログパレットコード(VRAM に書き込まれている値)，青，赤，緑の順で行います．レジスタとパレットの関係を図 I-4-22に示します．

▼表 I-4-6　アナログパレットレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FD90H	アナログパレットコード	R/W	P7*	P6*	P5*	P4*	P3	P2	P1	P0
FD92H	青色のパレットデータ	R/W	BL7	BL6	BL5	BL4	BL3*	BL2*	BL1*	BL0*
FD94H	赤色のパレットデータ	R/W	RL7	RL6	RL5	RL4	RL3*	RL2*	RL1*	RL0*
FD96H	緑色のパレットデータ	R/W	GL7	GL6	GL5	GL4	GL3*	GL2*	GL1*	GL0*

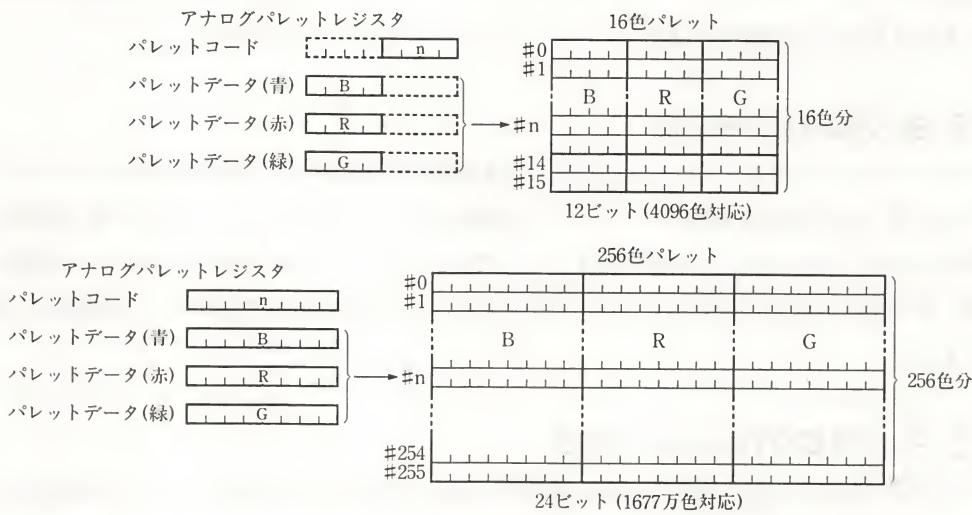
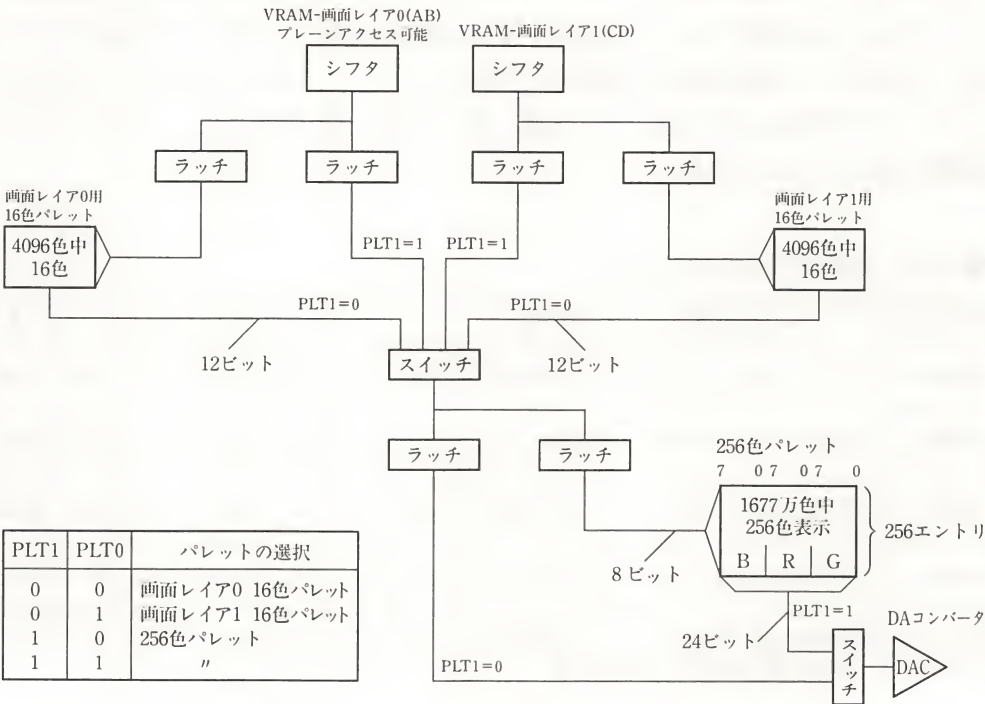
＊はビデオ出力制御(0448H，044AH，レジスタ番号01)のPLT1ビットが０の場合ライトでは０，リードでは不定．PLT1ビットが０の場合は全ビットR/W可能．

なお，アナログパレットコードから，表示する色への変換の過程を図示すると，図I-4-23のようになります．

この図中のPLT1は後述のプライオリティレジスタ(表I-4-36)のパレット選択ビットを指します．PLT1＝1のときは256色パレットを参照し，PLT1＝0のときは16色パレットを参照します．

▼図I-4-22　アナログパレットレジスタとパレットの関係

▼図I-4-23　アナログパレットの変換系統

PLT1	PLT0	パレットの選択
0	0	画面レイア0　16色パレット
0	1	画面レイア1　16色パレット
1	0	256色パレット
1	1	〃

なお，16色パレットは画面レイアごとに存在するので，画面合成時には最大31色(２つのパレットコードの透明を１色として数える)の表示が可能となります．

4.6 スプライト

この節では，スプライトを高速に表示，移動させる仕組みについて解説します．

4.6.1 スプライトの特徴

FMTOWNS のスプライトの仕様を表 I-4-7に示します．

スプライト１個のパターンの大きさは16×16ピクセルと小さいのですが，複数のスプライトを組み合わせることで，より大きなパターンを作ることができます．

また，１つのスプライトパターンをもとにして，表示の際にパターンを変形することもできます．これはハードウェアにより高速に行われています．

▼表 I-4-7　FMTOWNSのスプライトの仕様

項　目	仕　様
サイズ	16×16ピクセル
表示可能なスプライト数	1024個
使用できる色数	32768色，または32768色中16色(それぞれピクセル単位)
パターンの定義数	最大896(16色のスプライトだけを使った場合) 224(32768色のスプライトだけを使った場合)
パターンの重なり	優先順位処理が可能
表示時のパターンの変形	回転(０，90，180，270度) 左右反転 縮小(水平，垂直独立に１倍，1/2倍)

4.6.2 スプライトの表示

スプライトを表示する際には，VRAM に直接キャラクタのデータを書き込むわけではありません．

スプライトを表示する前の準備として，スプライトパターンメモリにスプライトのパターン(色と形)を定義しておきます(最大896個)．そして，スプライト(最大1024個)ごとに，スプライトのパターン番号，スプライトの表示位置，表示時の変形の有無などを定義します．

表示する際には，ハードウェアがスプライトの番号(インデックス番号)によりスプライトの種々のデータを読み出し，最後に VRAM に転送します．

スプライトの表示の仕組みを図 I-4-24に示します．

スプライトを表示する場合には，VRAMのページ1(画面レイア1)を仮想画面256×512ピクセルの画面モードで使用します．このページを，256×256ピクセルの大きさに2分割し，それぞれをスプライト表示用のバッファ(ダブルバッファ)として使用します．

背景の画面用には，VRAMのページ0を使用しますが，画面モードは仮想画面が256×512ピクセルでも512×256ピクセルでもかまいません．

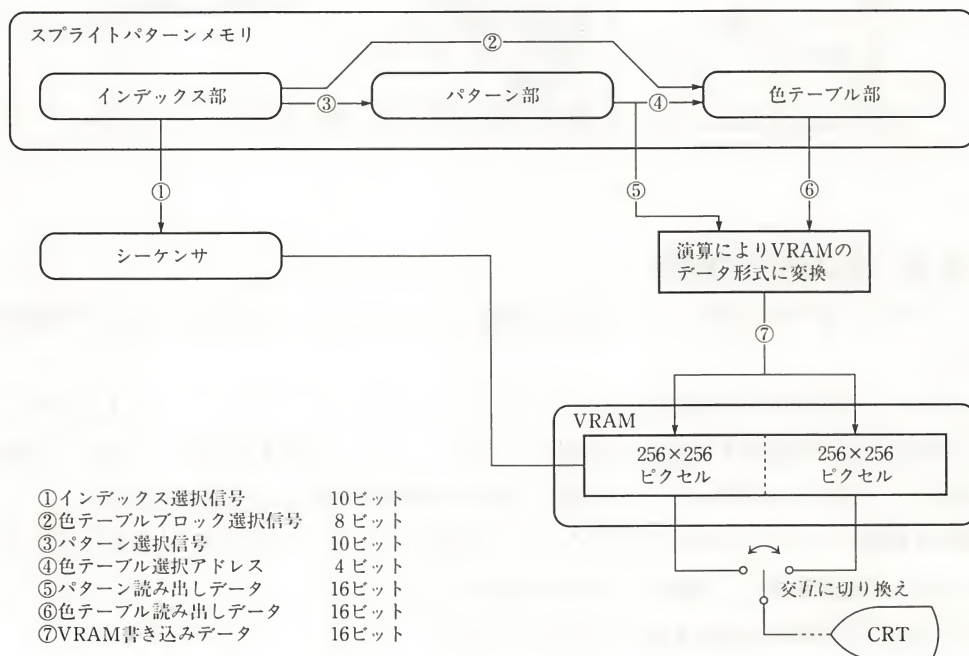
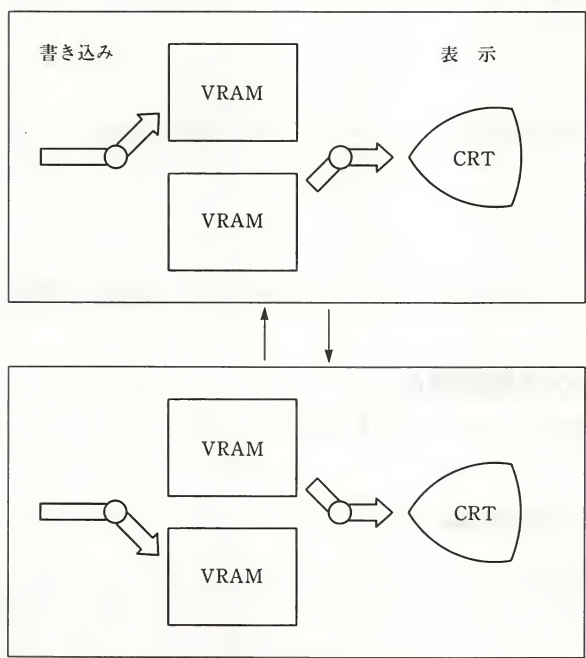
図Ⅰ-4-25に，ダブルバッファによるスプライトパターンの書き込みの関係を示します．

すなわちスプライトパターンメモリの内容を，1つのバッファに転送している間は，残りのバッファを表示し，転送が終わるとバッファを切り替えて，新しく転送したバッファを表示する，ということを繰り返します．

シングルバッファで表示を行うと，表示の途中でデータの転送が行われると，画面にノイズが入ったり，ちらつきが起きたりして見苦しい画面になります．また，これを避けるために，画面を表示(スキャン)している時間に転送を禁止し，表示しない時間(帰線区間)に転送すると時間待ちが必要になり，スプライトの書き込みが遅くなります．

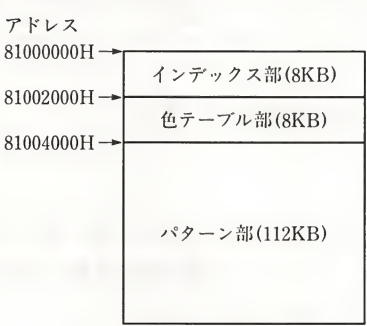
ダブルバッファを使用することにより，高速でノイズのないスプライト表示が可能になるわけです．

なお，スプライトを表示するページ1(画面レイア1)は，表示の優先順位を上にしておく必要があります．

▼図Ⅰ-4-24　スプライト表示のメカニズム

▼図 I-4-25　ダブルバッファによるスプライトの書き込み

4.6.3　スプライトパターンメモリの構造と働き

ここでは，スプライトパターンメモリの構造と働きについて説明します．

スプライトパターンメモリは図 I-4-26に示すように，3つの部分に分かれます．

▼図 I-4-26　スプライトパターンメモリの構成

●インデックス部

スプライト1個1個について，表示位置，パターン(形と色)などの属性，色テーブル番号(パレットの色)などの管理情報が格納されます．インデックス部にはスプライト1024個分の情報を格納できる領域(スプライト1個につき8バイト)があります．1024個のスプライトには0から1023までの番号を付けて区別します．これをインデックス番号といいます．インデックス番号

は，大きいほど先に処理されます．このため，表示位置が重なるときは，後で処理されるパターンが残り，表示されます．

●色テーブル部

16色のスプライトを使う場合に，インデックス部で設定する色テーブルのパターンを定義する領域です．

●パターン部

スプライトの各ピクセルの色をスプライトパターンとして定義する領域です．

4.6.4 インデックス部の構成

インデックス部の構成を，表I-4-8に示します．

▼表I-4-8　インデックス部の構成

インデックス番号	ワード0	1	2	3
#0	Xアドレス(16ビット)	Yアドレス(16ビット)	アトリビュート(16ビット)	色テーブル番号(16ビット)
#1	Xアドレス(16ビット)	Yアドレス(16ビット)	アトリビュート(16ビット)	色テーブル番号(16ビット)
#2	Xアドレス(16ビット)	Yアドレス(16ビット)	アトリビュート(16ビット)	色テーブル番号(16ビット)
⋮	⋮	⋮	⋮	⋮
#1022	Xアドレス(16ビット)	Yアドレス(16ビット)	アトリビュート(16ビット)	色テーブル番号(16ビット)
#1023	Xアドレス(16ビット)	Yアドレス(16ビット)	アトリビュート(16ビット)	色テーブル番号(16ビット)

インデックス部には，次のような内容を格納します．

●座標アドレス

座標アドレスのビット構成は，表I-4-9，表I-4-10のようになっています．ここには，スプライトを表示する座標を設定します．スプライト座標空間(X軸，Y軸方向とも0～511の範囲)の座標系で，スプライトパターンの左上角の位置を定義します．

▼表I-4-9　Xアドレス

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							SPX8	SPX7	SPX6	SPX5	SPX4	SPX3	SPX2	SPX1	SPX0

SPX8-0(bit8-0)　：スプライトのX軸方向のアドレスを表す．

▼表Ｉ-4-10　Ｙアドレス

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							SPY8	SPY7	SPY6	SPY5	SPY4	SPY3	SPY2	SPY1	SPY0

SPY8-0(bit8-0)　：スプライトのＹ軸方向のアドレスを表す．

●アトリビュート

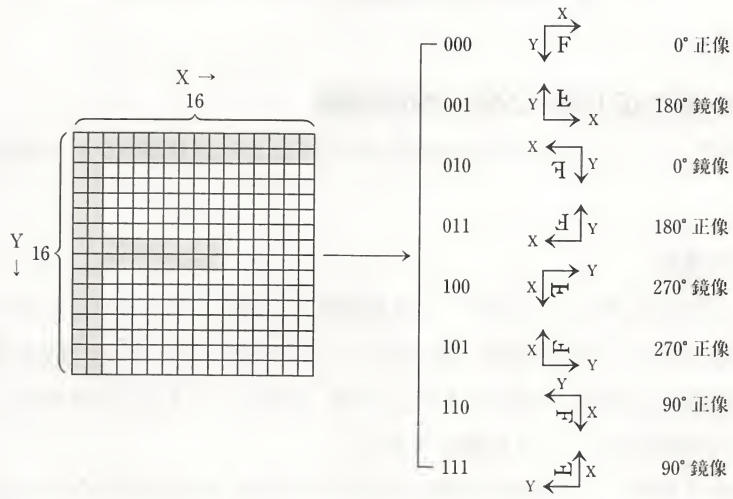
アトリビュートのビット構成は，表Ｉ-4-11のようになっています．

表示するスプライトのパターン番号の選択と，スプライトの表示の際の縮小(水平，垂直方向)，回転，左右反転などの有無，オフセットアドレスの加算の有無などを設定します．オフセットアドレスについては，「4.6.9　スプライトI/Oコントローラ」を参照してください．

▼表Ｉ-4-11　アトリビュート

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OFFS	ROT2	ROT1	ROT0	SUY	SUX	PAT9	PAT8	PAT7	PAT6	PAT5	PAT4	PAT3	PAT2	PAT1	PAT0

OFFS(bit15)　：オフセットアドレス参照．
　0＝参照しない
　1＝参照する

ROT2-0(bit14-12)　：回転/左右反転．

SUY(bit11)　：垂直縮小．
　0＝ノーマルサイズ
　1＝1/2縮小

SUX(bit10)　：水平縮小．
　0＝ノーマルサイズ
　1＝1/2縮小

PAT9-0(bit9-0)　：パターン番号．
　128～1023

●色テーブル番号

16色のスプライトを使うか，32768色のスプライトを使うかを設定すると同時に，16色のスプライトを使う場合の16色の組み合わせ(色テーブル)を選択するためのものです．

色テーブル番号部のビット構成は，表I-4-12のようになっています．色テーブル部に設定されている色パターンの組み合わせの番号を256～511で設定します．色テーブル番号部の最上位のCTENは，色テーブルを参照するかどうか，つまり，スプライトの色数を16色にするか32768色にするかどうかを決めるものです．

▼表I-4-12 色テーブル番号

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
CTEN	SPYS	DISP	/	COL11	COL10	COL9	COL8	COL7	COL6	COL5	COL4	COL3	COL2	COL1	COL0

CTEN(bit15)　：色テーブル参照．
0＝参照しない(32768色)
1＝参照する(16色)

SPYS(bit14)　：スプライトスーパーインポーズビット
0＝通常表示
1＝スーパーインポーズ

DISP(bit13)　：スプライト非表示フラグ
0＝表示
1＝非表示

COL11-0(bit11-0)：色テーブル番号．
256～511(その他は指定禁止)

4.6.5 パターン部へのパターンデータの格納

パターン部には，16色のパターンデータと32768色のパターンデータを混在させて格納することができます．

●16色のスプライトの場合

16色のスプライト1個のスプライトのパターンを格納するには，スプライトの1ピクセル当たりのメモリ容量が，4ビットであるため，16×16×4÷8＝128バイトのメモリが必要です．パターン部の全体の容量は112KB(114688バイト)なので，16色のスプライトだけを使った場合には，114688÷128＝896個のパターンを格納できます．

パターン部を先頭から128バイト単位の896個のブロックに分け，それぞれのブロックに896個のパターンデータを格納します．パターンの参照のために使われるパターン番号は，0から895番ではなく，128から1023までの番号になります．これは，スプライトパターンメモリの先頭から16KB(128個に相当)目までがインデックス部とパターン部なので，パターン番号としてこの値を使用すると，スプライトパターンメモリの先頭からの相対的な位置(アドレス)を計算するのに便利だからです．

●32768色のスプライトの場合

32768色のスプライトだけを使った場合も同様に考えることができます．

この場合には，16ビット／ピクセルなので，１つのスプライトのパターンを格納するには，512バイトのメモリが必要となります．したがって，224個のパターンを格納できることになります．パターン番号は，128～1020（4つおきに設定する）までです．

●両種のパターンを使う場合

両種のパターン番号の対応関係は，表Ｉ-4-13のようになります．

▼表Ｉ-4-13　16色と32768色のスプライト番号の対応

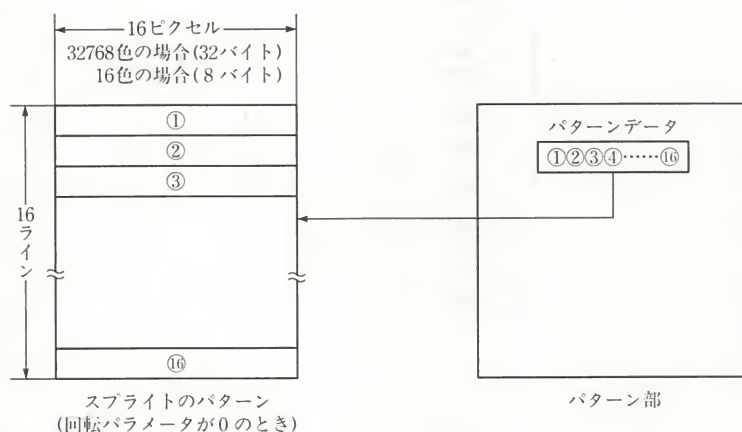
16色スプライト	32768色スプライト
#128	128
#129	
#130	
#131	
#132	132
#133	
#134	
#135	
#136	136
#137	
#138	
#139	
⋮	⋮
#1020	1020
#1021	
#1022	
#1023	

両種のスプライトは，パターン部として共通のメモリ領域を使用します．混在させて使用する場合には，パターン番号を調整して，同一のメモリを使用しないように注意してください．なお，参考までに両種のスプライトを混在させた場合のそれぞれの個数の例を表Ｉ-4-14に示します．

また，パターン部には，スプライトの上のラインから下のラインへの順で，パターンを格納します(図Ｉ-4-27)．

▼表 I-4-14　スプライトの組み合わせと合計個数の例

組み合わせ例	32768色スプライトパターン個数	32768色中16色スプライトパターン個数	スプライトパターン個数（合計）
1	224	0	224
2	192	128	320
3	160	256	416
4	128	384	512
5	96	512	608
6	64	640	704
7	0	896	896

▼図 I-4-27　パターン部のデータの並びとパターン

4.6.6 色テーブル部の構成

16色のスプライトでは，32768色の中から16色を選択して使用しますが，色テーブル部には，16色の組み合わせを256組定義することができます．

16色の色の組み合わせのパターンを1組分格納するには，16ビット(32768色)×16個÷8＝32バイトのメモリが必要ですから，256組のパターンを格納するには8KBのメモリを使用します．

256種類の組み合わせのパターンには，それぞれ番号(256〜511)をつけて区別します．これを色テーブル番号といいます．このように，0から255を使わない番号付けをしているのは，該当する色テーブルの先頭アドレスの計算が，スプライトパターンメモリの先頭アドレスからn×

32バイト(n は256～511で，色テー-ブル番号)で得られるようにするためです．

例えば，色テーブル番号が256の場合は，256×32＝8192となり，色テーブル部の先頭のアドレスを示します．

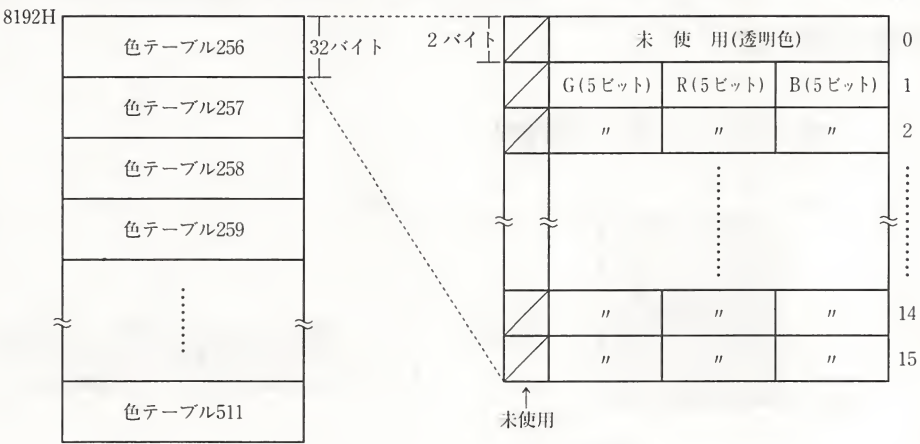
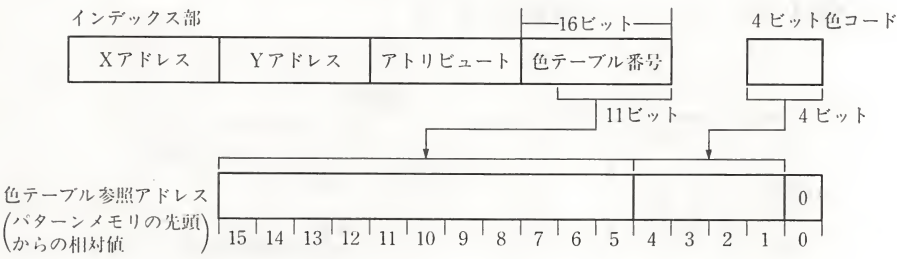
１組の色テーブルには，図Ｉ-4-28のように，16個の色番号に対してそれぞれ16ビット(32768色)のデータが格納されています．

16個の色番号から，それぞれの実際の色テーブルのアドレスの計算は，図Ｉ-4-29のように行います．インデックス部の色テーブル番号の下位11ビットが，色テーブル部の先頭アドレスを表しているわけです．

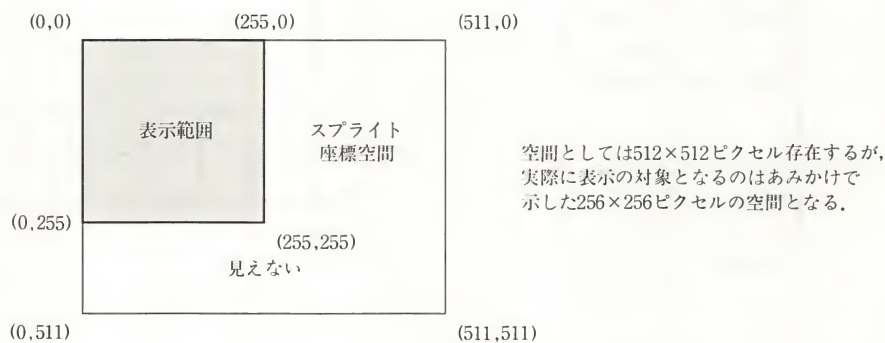
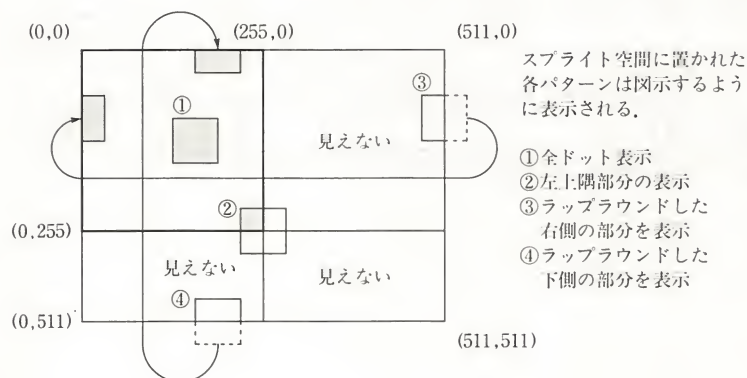
▼図Ｉ-4-28　色テーブル部のデータ格納形式

8192H
色テーブル256
色テーブル257
色テーブル258
色テーブル259
⋮
色テーブル511

32バイト

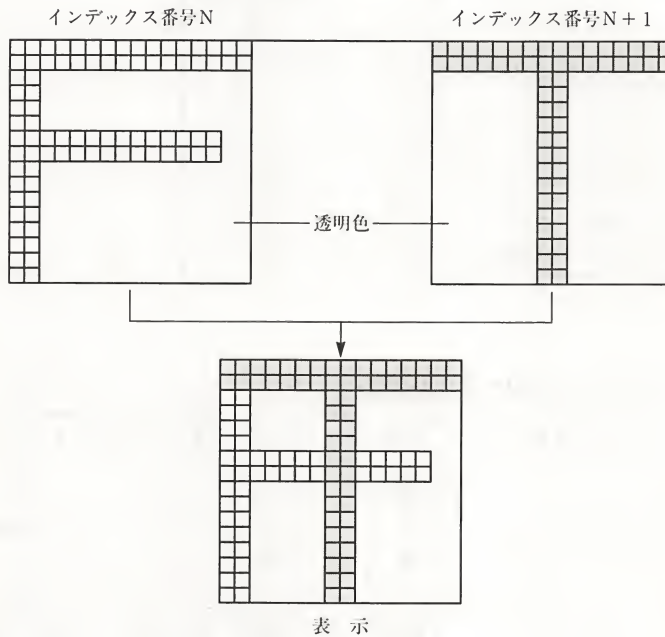
2バイト				
未使用	未　使　用(透明色)			0
未使用	G(5ビット)	R(5ビット)	B(5ビット)	1
未使用	〃	〃	〃	2
	⋮			⋮
未使用	〃	〃	〃	14
未使用	〃	〃	〃	15

▼図Ｉ-4-29　色テーブル番号と色テーブルのアドレス

4.6.7　スプライトの座標空間と表示範囲

スプライトの表示には，ページ1の256×512ピクセルの仮想画面を2つに分けて使用するので，スプライトを表示できる空間は，256×256ピクセルということになります．しかし，この空間はスプライトの移動空間としては狭すぎます．そこで，図Ⅰ-4-30のように，スプライトの論理的な座標空間を，512×512ピクセルの範囲として，スプライトパターンメモリのアトリビュート部に0から511の値を指定できるようにしています．ただし，実際にスプライトが表示されるのは，斜線で示された領域だけです．残りの部分は，論理空間としては存在していますが，物理的には意味がありません．しかし，このような座標空間を使うことにより，スプライトの論理的位置を変えるだけで，スプライトの一部または全部を消すことも可能です．さらに，図Ⅰ-4-31のように，512×512ピクセルの空間において，スプライトは画面の球面スクロールと同様な移動が可能です．

▼図Ⅰ-4-30　スプライト座標空間

▼図Ⅰ-4-31　スプライトの球面スクロール

4.6.8　優先順位とマスク処理

1024 個のスプライトは，インデックス番号の小さい方から処理されます．このため，スプライトを重ねて表示させた場合，インデックス番号の小さいものほど背後に表示されることになります（図 I-4-32）．

ただし，マスク処理を使って，インデックス番号の小さいスプライトのピクセルを表示させることもできます．

▼図 I-4-32　スプライトの重ね合わせ

●32768色の場合

パターン部に格納されている１ピクセルの 16 ビットのうち，最上位の１ビットが，マスク用として使われます．このビットが１の場合には透明色と解釈され，インデックス番号の小さいスプライトのピクセルが表示されます．

●16色の場合

１ピクセルの４ビットがすべて０のときは，透明色と解釈され，インデックス番号の小さいスプライトのピクセルが表示されます．

4.6.9 スプライトI/Oコントローラ

スプライトの制御には，スプライトI/Oコントローラを使用します．スプライトI/Oコントローラには，表I-4-15に示すような内部レジスタがあります．

内部レジスタは，スプライトコントローラI/Oレジスタ(表I-4-16)から間接的にアクセスします．内部レジスタへ書き込みを行う場合には，まず，0450H番地にレジスタ番号を書いておき，続けて0452H番地に値を書きます．

▼表I-4-15 スプライトI/Oコントローラの内部レジスタ

レジスタ番号	レ ジ ス タ 名	Word/Byte
0 0	コントロールレジスタ0	B
0 1	コントロールレジスタ1	B
0 2	水平オフセットレジスタ0	B
0 3	水平オフセットレジスタ1	B
0 4	垂直オフセットレジスタ0	B
0 5	垂直オフセットレジスタ1	B
0 6	表示ページ制御レジスタ	B

▼表I-4-16 スプライトコントローラI/Oレジスタ

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0450H</td><td rowspan="2">アドレスレジスタ</td><td>R</td><td colspan="5">不 定</td><td rowspan="2">RA2</td><td rowspan="2">RA1</td><td rowspan="2">RA0</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>0452H</td><td>データレジスタ</td><td>R/W</td><td>RD7</td><td>RD6</td><td>RD5</td><td>RD4</td><td>RD3</td><td>RD2</td><td>RD1</td><td>RD0</td></tr>
</table>

●コントロールレジスタ0，1

コントロールレジスタ0，1（表I-4-17）のIND9-0までの10ビットでは，制御するスプライトの個数（1～1024）を2の補数で設定します．なお，1個だけ表示したい場合は，インデックス番号#1023にデータを設定し，レジスタの設定値（IND9-0）を3FFhにする．

SPENはスプライト転送（スプライトパターンメモリから，VRAMへの転送）を行うかどうかを指定します．

▼表Ⅰ-4-17　コントロールレジスタ

レジスタ番号	内　　容	7	6	5	4	3	2	1	0
0 0	コントロールレジスタ 0	IND7	IND6	IND5	IND4	IND3	IND2	IND1	IND0
0 1	コントロールレジスタ 1	SPEN						IND9	IND8

SPEN (bit7)　：スプライトパターンの転送動作を行うか否かの指定.
0＝スプライトパターンの転送動作を行わない
1＝スプライトパターンの転送動作を実行する
（リセット時はスプライトパターンの転送動作を行わない状態となる）

IND9-0　：制御するスプライトの個数（1～1024）を２の補数で指定する.
(0を指定した場合に1024個すべての制御)

●水平垂直オフセットレジスタ０，１

水平垂直オフセットレジスタ０，１(表Ⅰ-4-18)は，スプライトの表示座標のオフセット値を設定するものです．スプライトパターンメモリのインデックス部の座標値は，スプライトごとに設定しますが，このオフセット値は，それを参照するすべてのスプライトに対して，表示位置のオフセット値となります．

OY8-0までの９ビットで，垂直方向のオフセットを設定し，OX8-0までの９ビットで水平方向のオフセットを設定します．

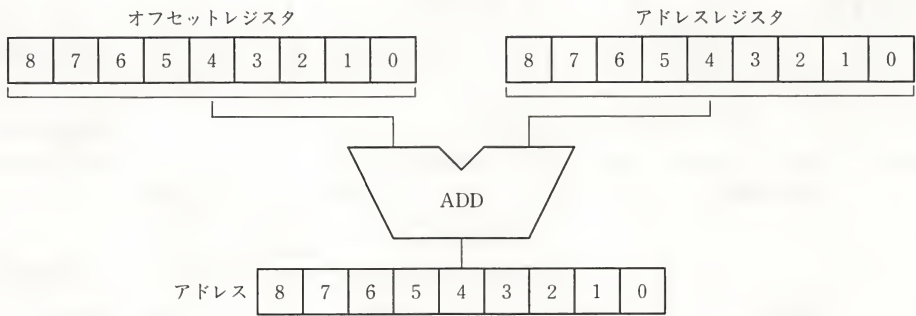
このオフセット値は，インデックス部のアトリビュートのOFFSの値が１の場合にのみ加算が行われます．オフセット値とインデックス部の座標値の加算は，図Ⅰ-4-33のように行われます．

そして，加算した値が８ビットを超えた場合は，繰り上がりは無視されるので，大きな値を加えても，表示座標は０～511の範囲にあることになり，スプライト群を球面スクロールのような形態で移動させることができます．

▼表Ⅰ-4-18　水平垂直オフセットレジスタ

レジスタ番号	内　　容	7	6	5	4	3	2	1	0
0 2	水平オフセットレジスタ 0	OX7	OX6	OX5	OX4	OX3	OX2	OX1	OX0
0 3	水平オフセットレジスタ 1								OX8
0 4	垂直オフセットレジスタ 0	OY7	OY6	OY5	OY4	OY3	OY2	OY1	OY0
0 5	垂直オフセットレジスタ 1								OY8

水平，垂直とも９ビットでオフセット値を構成.

▼図Ⅰ-4-33　オフセットレジスタ値のアダレス値への加算（XまたはY方向）

●表示ページ制御レジスタ

表示ページ制御レジスタ（表Ⅰ-4-19）は，DP1だけに意味があります．

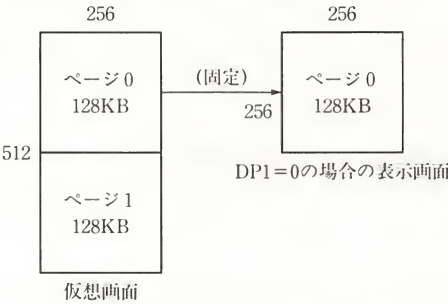
スプライトは，VRAMの画面レイア1をダブルバッファとして使っています．そしてある瞬間においてはダブルバッファのどちらかが表示されています．DP1は，スプライト停止時にそのどちらを表示するかを示すものです．

このレジスタによる書き込みに意味があるのは，図Ⅰ-4-34のように表示ページを選択する場合です．DP1に0を書き込むとこの図のページ0が，1を書き込むとページ1が表示されま

▼表Ⅰ-4-19　表示ページ制御レジスタ

レジスタ番号	内　　容	R/W	7	6	5	4	3	2	1	0
0 6	表示ページ制御レジスタ	R	0	0	0	DP1	0	0	0	0
		W	DP1	0	0	0	0	0	0	0

DP1　：表示ページ選択．
　0＝ページ0
　1＝ページ1

▼図Ⅰ-4-34　スプライトの固定表示

す．したがって，書き込みを行う場合には，CRTCの表示タイミングをVRAMのページ0が表示されるように設定しておいて，スプライトパターンの転送を停止するために前述のコントロールレジスタ1（表I-4-17）のSPENを0（転送しない）にしておく必要があります．なお，表示ページ制御レジスタは，スプライトが使える256×512ピクセルの仮想画面以外の仮想画面では意味がありません．それらの場合にはDP1は常に0に設定しておく必要があります．

ここでは，ダブルバッファ（VRAMの画面レイア1）の前半をページ0，後半部をページ1と呼んでいるので，注意してください．

4.7　CRTC周辺のハードウェアの仕組み

CRTC(Cathode Ray Tube Controller)は画面表示を制御するLSIです．

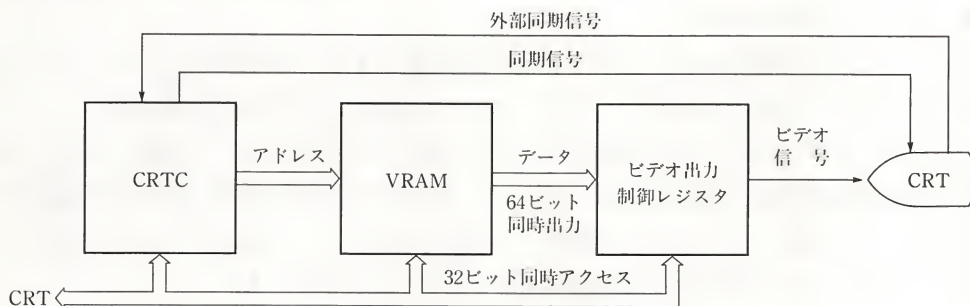
この節では，CRTC周辺のハードウェアについて説明します．

4.7.1　CRTC周辺の概要

CRTC周辺のハードウェアの概略を図I-4-35に示します．

CRTCは，VRAMのデータをディスプレイに表示させるために重要な役割を果しています．すなわち，CRTCのレジスタへの書き込みを行うことによって，表示のタイミングなどを制御しており，VRAMにデータをいくら書き込んでも，CRTCのレジスタに対して適切な設定を行わなければ，正常な画面表示はできません．

また，CRTC周辺には，ビデオ出力制御部のレジスタがあり，各種の制御を行っています．

▼図I-4-35　CRTC周辺の概略図

4.7.2 ブラウン管の表示の仕組み

CTRCによる画面の制御の解説をする前に，その前提の知識として，ブラウン管の表示の仕組みについて簡単に説明します．

●ラスタスキャン

パソコンやテレビ放送のディスプレイに採用されているブラウン管の表示方法をラスタスキャン方式といいます．

ブラウン管の表示動作を瞬間的に捉えると，実際に光を発しているのは，ただ1点にすぎません．この点をスポットといいます．

スポットは水平方向に移動し，ラスタ(輝線，または走査線ともいう)を描きます．さらに，ラスタは画面の上から下へ向かって順次移動し，最下端に達すると，また最上端から同じ動作を繰り返します．このような動作をスキャン(走査)といいます(図Ⅰ-4-36)．

1画面は，最上端のラスタから最下端のラスタまでとなりますが，単位時間当たりの画面の表示回数が少ないとちらつきが目立ち，目が疲れやすくなります．だからといって，表示回数を増やすと，単位時間当たりのデータ転送量を増やす必要が生じ，ハードウェアの高速化が必要になります．

テレビ放送の規格では，1画面の走査線の数は486本となっており，1画面の書き換えに30分の1秒を要します．しかし，この書き換えの速度では，ちらつきが感じられるので，1回の表示を，奇数ラスタと偶数ラスタ(それぞれ243本)にわけ，それぞれを60分の1秒単位で表示することによって，ちらつきを少なくする方法が採用されています．これをインタレース方式といいます．

コンピュータの画面の走査線の数は，機種や，画面のモードなどで異なりますが，FMTOWNSの走査線の数は，表示画面の縦の解像度に対応して，選択できるようになっています．

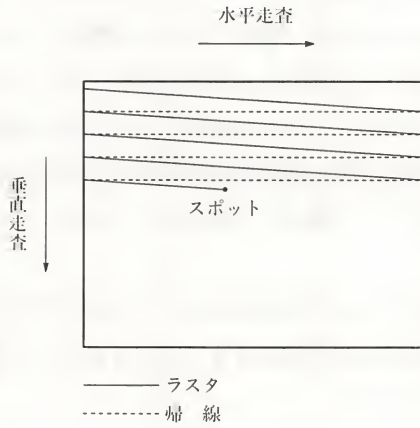
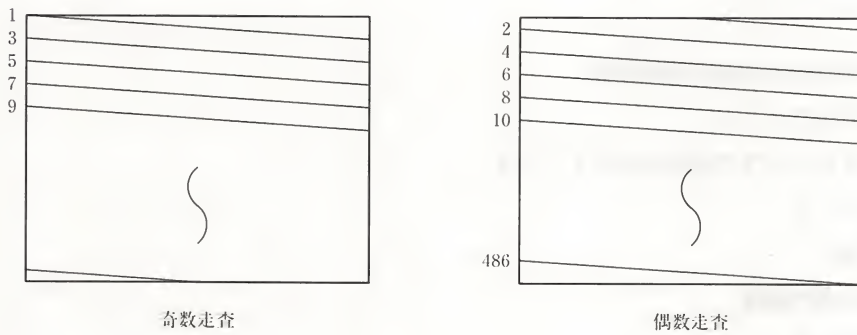
●インタレースと等化パルス

インタレースは，前述のように一般のテレビで使われており，FMTOWNSでもインタレースの表示が可能です．図Ⅰ-4-37にテレビのインタレースのラスタの描き方を示します．

この図のように，奇数ラスタの最終のラスタと偶数ラスタの最初のラスタはいずれも，水平方向の表示時間の半分しかありません．そこで，タイミングを合わせるために水平同期信号の代わりに，水平同期パルスの半分の間隔のものを挿入します．これを等化パルスといいます．等化パルスのタイミングの設定は垂直同期を設定するレジスタで行います．

●水平同期信号と垂直同期信号

ブラウン管に表示を行う際には，ビデオ信号の外に，水平と垂直のスキャンのタイミングを知らせる水平同期信号と垂直同期信号が必要です．

▼図 I-4-36　ラスタスキャンの概念

▼図 I-4-37　テレビのインタレース表示

水平同期信号は，走査線のスキャンの開始を指示する信号です．ディスプレイ装置はこの信号を受け取ると水平方向のスキャンを止め，スポットは画面の左端にもどります．水平同期信号の周期は，水平周波数によって決まり，同期に要する時間を水平同期期間といいます．

また，垂直同期信号は，１画面のスキャンの開始を指示する信号です．ディスプレイ装置はこの信号を受け取ると垂直方向のスキャンを止め，スポットは画面の最上位にもどります．垂直同期の周期は，水平周期に走査線の数を掛けたものと，スポットが画面の最上位にもどるまでの垂直同期期間を加えたものとなります．そして，垂直同期期間には水平方向のスキャンを何回も行えるだけの時間がかかります．この期間には画面表示は行われませんが，水平同期信号はひきつづき出力されるようになっています．これは，水平同期信号を一度止めるとディスプレイ側で再同期するのに少し時間がかかるためです．

FMTOWNS では，15.73KHz，24.37KHz，31.47KHz などの水平周波数を選択できます．

●アンダースキャンとオーバースキャン

表示画面の範囲が，ディスプレイ画面の大きさより小さい場合を，アンダースキャン，表示画面の周辺部が，ディスプレイからあふれて表示されない状態のことをオーバースキャンといいます．この違いは，走査線の数と表示画面の解像度の関係によって生じます．

両者について2つの例で説明します．まず，画面モード1(640×400ピクセル)の場合，FM TOWNSのディスプレイの走査線の数は440本(実際に表示できる走査線の数は432本)に設定します．すると，432本の走査線のうち400本しか使われないため，アンダースキャンとなります．

また，画面モード11(320×240ピクセル)の場合には，ディスプレイの走査線の数は262本(実際に表示できる走査線の数は216本）に設定します．すべてのラインを表示することはできなくなり，オーバースキャンになります．

4.7.3　CRTCのレジスタとその設定例

CRTCの内部のレジスタに値を書き込むことにより，ディスプレイ表示に関して次のような項目の設定ができます．

水平同期周期と垂直同期周期
表示画面のサイズ
VRAMのなかでの実際の表示アドレス
スクロール
拡大表示
仮想画面の種類
同時発色数

この中で，最も重要なのは，表示するVRAMのアドレスを生成することです．CRTCはスキャンに合わせて，VRAMのアドレスを更新します．

CRTCには，たくさんの内部レジスタがあり，画面表示の制御を行います．その内訳は表Ⅰ-4-20のとおりです．

これらのレジスタには，それぞれ独立して値を個別に設定できますが，32個のすべてのレジスタの働きを完全に理解して適切な設定をしなければ，画面表示が乱れることになります．

そこで，CRTCの内部レジスタの標準設定値として，画面モードと各レジスタの値の組み合わせの一覧を表Ⅰ-4-21と表Ⅰ-4-22に示します．表中のレジスタセット番号は，複数のレジスタの設定値の組み合わせを示す番号です．レジスタセット番号と各レジスタの設定値の関係は表Ⅰ-4-23に示すとおりです．

2画面を使用する場合には，画面レイアの割り当てによって，レジスタの設定値が異なるので注意してください．

表Ⅰ-4-23には，ビデオ出力制御レジスタの設定値も含まれています．これについては，「4.8　ビデオ出力制御部と関連レジスタ」を参照してください．

なお，スーパーインポーズやデジタイズなどを行う場合には，それに合わせて各レジスタの設定値の変更が必要です．これについては，「4.11　ビデオボード」を参照してください．

▼表Ⅰ-4-20　CRTC内部レジスタ一覧

レジスタ番号	略称	レジスタ名	用途	サイズ
00	HSW1	水平同期幅1	水平同期信号設定	W
01	HSW2	水平同期幅2		W
02		予約済		
03		予約済		
04	HST	水平同期周期	水平同期周期設定	W
05	VST1	垂直同期時間1	垂直同期信号設定	W
06	VST2	垂直同期時間2		W
07	EET	等化パルス有効時間	等化パルス波形設定	W
08	VST	垂直同期周期	垂直同期周期設定	W
09	HDS0	水平表示開始位置0	画面レイア0水平表示位置設定	W
0A	HDE0	水平表示終了位置0		W
0B	HDS1	水平表示開始位置1	画面レイア1水平表示位置設定	W
0C	HDE1	水平表示終了位置1		W
0D	VDS0	垂直表示開始位置0	画面レイア0垂直表示位置設定	W
0E	VDE0	垂直表示終了位置0		W
0F	VDS1	垂直表示開始位置1	画面レイア1垂直表示位置設定	W
10	VDE1	垂直表示終了位置1		W
11	FA0	フレーム先頭アドレス0	画面レイア0スクロール設定	W
12	HAJ0	水平アジャスト0		W
13	FO0	フィールド間アドレスオフセット0	画面レイア0インタレース表示設定	W
14	LO0	ライン間アドレスオフセット0		W
15	FA1	フレーム先頭アドレス1	画面レイア1スクロール設定	W
16	HAJ1	水平アジャスト1		W
17	FO1	フィールド間アドレスオフセット1	画面レイア1インタレース表示設定	W
18	LO1	ライン間アドレスオフセット1		W
19	EHAJ	外部同期水平アジャスト	外部同期位置合わせ設定	W
1A	EVAJ	外部同期垂直アジャスト		W
1B	ZOOM	水平垂直拡大	水平垂直拡大率設定	W
1C	CR0	コントロールレジスタ0	各種コントロール	W
1D	CR1	コントロールレジスタ1		W
1E	FR	ダミーレジスタ	ダミーレジスタ	W
1F	CR2	コントロールレジスタ2	垂直同期信号分離回路用のレジスタ	W

▼表I-4-21　1画面モードの場合の画面モードとレジスタセット番号の関係

画面モード	レジスタセット番号
12	1
15	4
17	31
13	2
14	3
16	5
18	32

▼表I-4-22　2画面モードの場合の画面モードとレジスタセット番号の関係

	画面レイア0											
	画面モード	1	2	3	4	5	6	7	8	9	10	11
画面レイア1	1	6										
	2		8									
	3			9		19					27	
	4	R50互換			10		11					
	5			22		20					29	
	6				23		16					
	7							17		24		
	8								18			25
	9							13		12		
	10			26		30					28	
	11								15			14

▼表 I-4-23　レジスタ設定値

レジスタアドレス	レジスタ名	レジスタセット番号									
		1	2	3	4	5	6	7	8	9	10
0 0	HSW1	0060	0040	0086	0060	0074	0040		0040	0060	0040
0 1	HSW2	02C0	0320	0610	02C0	0530	0320		0320	02C0	0320
0 2	――	――	――	――	――	――	――	――	――	――	――
0 3	――	――	――	――	――	――	――	――	――	――	――
0 4	HST	031F	035F	071B	031F	0617	035F	未使用	035F	031F	035F
0 5	VST1	0000	0000	0006	0000	0006	0000		0000	0000	0000
0 6	VST2	0004	0010	000C	0004	000C	0010		0010	0004	0010
0 7	EET	0000	0000	0012	0000	0012	0000		0000	0000	0000
0 8	VST	0419	036F	020C	0419	020C	036F	予約済	036F	0419	036F
0 9	HDS0	008A	009C	0129	008A	00E7	009C		009C	008A	009C
0 A	HDE0	030A	031C	06C9	030A	05E7	031C		031C	030A	031C
0 B	HDS1	008A	009C	0129	008A	00E7	009C		009C	008A	009C
0 C	HDE1	030A	031C	06C9	030A	05E7	031C		031C	030A	031C
0 D	VDS0	0046	0040	002A	0046	002A	0040		0040	0046	0040
0 E	VDE0	0406	0360	020A	0406	020A	0360		0360	0406	0360
0 F	VDS1	0046	0040	002A	0046	002A	0040		0040	0046	0040
1 0	VDE1	0406	0360	020A	0406	020A	0360		0360	0406	0360
1 1	FA0	0000	0000	0000	0000	0000	0000		0000	0000	0000
1 2	HAJ0	008A	009C	0129	008A	00E7	009C		009C	008A	009C
1 3	FO0	0000	0000	0080	0000	0080	0000		0000	0000	0000
1 4	LO0	0080	0080	0100	0080	0100	0050		0050	0080	0080
1 5	FA1	0000	0000	0000	0000	0000	0000		0000	0000	0000
1 6	HAJ1	008A	009C	0129	008A	00E7	009C		009C	008A	009C
1 7	FO1	0000	0000	0080	0000	0080	0000		0000	0000	0000
1 8	LO1	0080	0080	0100	0080	0100	0050		0050	0080	0080
1 9	EHAJ	0058	004A	0064	0058	0056	004A		004A	0058	004A
1 A	EVAJ	0001	0001	0007	0001	0007	0001		0001	0001	0001
1 B	ZOOM	0000	0000	0101	0101	0303	0000		1010	0000	0000
1 C	CR0	000F	000F	000F	000A	000A	003F		003F	000F	000F
1 D	CR1	0002	0003	000C	0002	0001	0003		0003	0002	0003
1 E	FR	0000	0000	0003	0000	0002	0000		0000	0000	0000
1 F	CR2	0192	0150	01CA	0192	0188	0150		0150	0192	0150

SIFTER（ビデオ出力制御レジスタ）

0 0	コントロールレジスタ	0A	0A	0A	0F	0F	15		15	15	15
0 1	プライオリティレジスタ	18	18	18	08	08	08		08	08	08

レジスタアドレス	レジスタ名	レジスタセット番号									
		11	12	13	14	15	16	17	18	19	20
0 0	HSW1	0040	0086	0086	0074	0074	0040	0086	0074	0060	0060
0 1	HSW2	0320	0610	0610	0530	0530	0320	0610	0530	02C0	02C0
0 2	―――	――	――	――	――	――	――	――	――	――	――
0 3	―――	――	――	――	――	――	――	――	――	――	――
0 4	HST	035F	071B	071B	0617	0617	035F	071B	0617	031F	031F
0 5	VST1	0000	0006	0006	0006	0006	0000	0006	0006	0000	0000
0 6	VST2	0010	000C	000C	000C	000C	0010	000C	000C	0004	0004
0 7	EET	0000	0012	0012	0012	0012	0000	0012	0012	0000	0000
0 8	VST	036F	020C	020C	020B	020B	036F	020C	020B	0419	0419
0 9	HDS0	009C	0129	0129	00E7	00E7	009C	0129	00E7	008A	008A
0 A	HDE0	019C	06C9	0529	05E7	04E7	019C	0529	04E7	018A	018A
0 B	HDS1	009C	0129	0129	00E7	00E7	009C	0129	00E7	008A	008A
0 C	HDE1	031C	06C9	06C9	05E7	05E7	019C	0529	04E7	030A	018A
0 D	VDS0	0040	002A	002A	002A	002A	0040	002A	002A	0046	0046
0 E	VDE0	0240	020A	020A	020A	020A	0240	020A	020A	0246	0246
0 F	VDS1	0040	002A	002A	002A	002A	0040	002A	002A	0046	0046
1 0	VDE1	0360	020A	020A	020A	020A	0240	020A	020A	0406	0246
1 1	FA0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1 2	HAJ0	009C	0129	0129	00E7	00E7	009C	0129	00E7	008A	008A
1 3	FO0	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1 4	LO0	0080	0100	0080	0100	0080	0080	0080	0080	0080	0080
1 5	FA1	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1 6	HAJ1	009C	0129	0129	00E7	00E7	009C	0129	00E7	008A	008A
1 7	FO1	0000	0000	0000	0000	0000	0000	0000	0000	0000	0000
1 8	LO1	0080	0100	0100	0100	0100	0080	0080	0080	0080	0080
1 9	EHAJ	004A	0064	0064	0056	0056	004A	0064	0056	0058	0058
1 A	EVAJ	0001	0007	0007	0007	0007	0001	0007	0007	0001	0001
1 B	ZOOM	0000	0303	0303	0303	0303	0000	0303	0303	0000	0000
1 C	CR0	000D	0005	0005	0005	0005	0005	0005	0005	000D	0005
1 D	CR1	0003	000C	000C	0001	0001	0003	000C	0001	0002	0002
1 E	FR	0000	0003	0003	0002	0002	0000	0003	0002	0000	0000
1 F	CR2	0150	01CA	01CA	0188	0188	0150	01CA	0188	0192	0192

SIFTER（ビデオ出力制御レジスタ）

0 0	コントロールレジスタ	17	1F	1F	1F	1F	1F	1F	1F	17	1F
0 1	プライオリティレジスタ	08	08	08	08	08	08	08	08	08	08

レジスタアドレス	レジスタ名	レジスタセット番号								
		21	22	23	24	25	26	27	28	29	30
0 0	HSW1	未使用	0060	0040	0086	0074	0060	0060	0060	0060	0060
0 1	HSW2		02C0	0320	0610	0530	02C0	02C0	02C0	02C0	02C0
0 2	――	――	――	――	――	――	――	――	――	――	――
0 3	――	――	――	――	――	――	――	――	――	――	――
0 4	HST		031F	035F	071B	0617	031F	031F	031F	031F	031F
0 5	VST1		0000	0000	0006	0006	0000	0000	0000	0000	0000
0 6	VST2		0004	0010	000C	000C	0004	0004	0004	0004	0004
0 7	EET		0000	0000	0012	0012	0000	0000	0000	0000	0000
0 8	VST	予約済	0419	036F	020C	020B	0419	0419	0419	0419	0419
0 9	HDS0		008A	009C	0129	00E7	008A	008A	008A	008A	008A
0 A	HDE0		030A	031C	06C9	05E7	030A	01CA	01CA	01CA	01CA
0 B	HDS1		008A	009C	0129	00E7	008A	008A	008A	008A	008A
0 C	HDE1		018A	019C	0529	04E7	01CA	030A	01CA	018A	018A
0 D	VDS0		0046	0040	002A	002A	0046	0046	0046	0046	0046
0 E	VDE0		0406	0360	020A	020A	0406	0226	0226	0226	01E6
0 F	VDS1		0046	0040	002A	002A	0046	0046	0046	0046	0046
1 0	VDE1		0246	0240	020A	020A	0226	0406	0226	0246	0226
1 1	FA0		0000	0000	0000	0000	0000	0000	0000	0000	0000
1 2	HAJ0		008A	009C	0129	00E7	008A	008A	008A	008A	008A
1 3	FO0		0000	0000	0000	0000	0000	0000	0000	0000	0000
1 4	LO0		0080	0080	0100	0100	0080	0100	0100	0100	0080
1 5	FA1		0000	0000	0000	0000	0000	0000	0000	0000	0000
1 6	HAJ1		008A	009C	0129	00E7	008A	008A	008A	008A	008A
1 7	FO1		0000	0000	0000	0000	0000	0000	0000	0000	0000
1 8	LO1		0080	0080	0080	0080	0100	0080	0100	0080	0100
1 9	EHAJ		0058	004A	0064	0056	0058	0058	0058	0058	0058
1 A	EVAJ		0001	0001	0007	0007	0000	0000	0000	0000	0000
1 B	ZOOM		0000	0000	0303	0303	0000	0000	0000	0000	0000
1 C	CR0		0007	0007	0005	0005	0007	000D	0005	0005	0005
1 D	CR1		0002	0003	000C	0001	0002	0002	0002	0002	0002
1 E	FR		0000	0000	0003	0002	0000	0000	0000	0000	0000
1 F	CR2		0192	0150	01CA	0188	0192	0192	0192	0192	0192

SIFTER（ビデオ出力制御レジスタ）

0 0	コントロールレジスタ		1D	1D	1F	1F	1D	17	1F	1F	1F
0 1	プライオリティレジスタ		08	08	08	08	08	08	08	08	08

レジスタアドレス	レジスタ名	レジスタセット番号		
		31	32	R50
0 0	HSW1	0060	0074	0040
0 1	HSW2	02C0	0530	0320
0 2	――――	――	――	――
0 3	――――	――	――	――
0 4	HST	031F	0617	035F
0 5	VST1	0000	0006	0000
0 6	VST2	0004	000C	0010
0 7	EET	0000	0012	0000
0 8	VST	0419	020C	036F
0 9	HDS0	00CA	0167	009C
0 A	HDE0	02CA	0567	031C
0 B	HDS1	00CA	0167	009C
0 C	HDE1	02CA	0567	031C
0 D	VDS0	0046	002A	0040
0 E	VDE0	0406	020A	0360
0 F	VDS1	0046	002A	0040
1 0	VDE1	0406	020A	0360
1 1	FA0	0000	0000	0000
1 2	HAJ0	00CA	0167	009C
1 3	FO0	0000	0080	0000
1 4	LO0	0080	0100	0050
1 5	FA1	0000	0000	0000
1 6	HAJ1	00CA	0167	009C
1 7	FO1	0000	0080	0000
1 8	LO1	0080	0100	0080
1 9	EHAJ	0058	0056	004A
1 A	EVAJ	0001	0001	0001
1 B	ZOOM	0000	0101	0000
1 C	CR0	000A	000A	001F
1 D	CR1	0002	0001	0003
1 E	FR	0000	0002	0000
1 F	CR2	0192	0188	0150

SIFTER(ビデオ出力制御レジスタ)

0 0	コントロールレジスタ	0F	0F	15
0 1	プライオリティレジスタ	08	08	09

外部同期時の注意事項

32768色　1画面モード
　スーパーインポーズ時　：画面レイア1をDISPLAY OFFにしておく.
　　ONの場合スーパーインポーズしない.
　デジタイズ時　：画面レイア1はDISPLAY ONにしておく.
　　OFFの場合デジタイズスルーが出ない.

256色　1画面モード
　スーパーインポーズ時　：Ysの有効／無効の切り換えはできない.
　　常に有効.

4.7.4 CRTC の内部レジスタ

ここでは，CRTC の内部のレジスタの個々の働きについて説明します．

● CRTC の内部レジスタへの書き込みの手順

CRTC の内部レジスタを読み書きするには，CRTC の I/O レジスタを使用します（表 I - 4 - 24）．

0440H 番地に，読み書きを行う内部レジスタの番号を設定し，続けて，0442H 番地と0043H 番地に，書き込むデータそのものを設定します．

なお，FMTOWNS に使われている CRTC はインテル系の CPU を意識して設計されているので，ワード（16ビット）データを転送する命令を使用して，CRTC の内部レジスタを読み書きすることができます．

▼表 I-4-24　CRTCのI/Oレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0440H	アドレスレジスタ	W	0	0	0	RA4	RA3	RA2	RA1	RA0
0442H	データレジスタ（下位）	W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
0443H	データレジスタ（上位）	W	RD15	RD14	RD13	RD12	RD11	RD10	RD9	RD8

●同期信号関係のレジスタ

同期信号関係の各レジスタの形式を，表 I -4-25に示します．

▼表 I-4-25　同期信号関係のレジスタ

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0 0	水平同期幅 1 レジスタ						0	0	0	HSW1							0
0 1	水平同期幅 2 レジスタ						0	0	0	HSW2							0
0 4	水平同期周期レジスタ						HST										1
0 5	垂直同期時間 1 レジスタ						0	0	0	0	0	0	VST1				
0 6	垂直同期時間 2 レジスタ						0	0	0	0	0	0	VST2				
0 7	等化パルス有効時間レジスタ						0	0	0	0	0	0	EET				
0 8	垂直同期周期レジスタ						VST										

●水平同期信号関係のレジスタ

水平同期信号の周期とレジスタの設定値の関係を，図Ⅰ-4-38に示します．

全体の周期は，水平同期周期レジスタ(HST)に設定します．

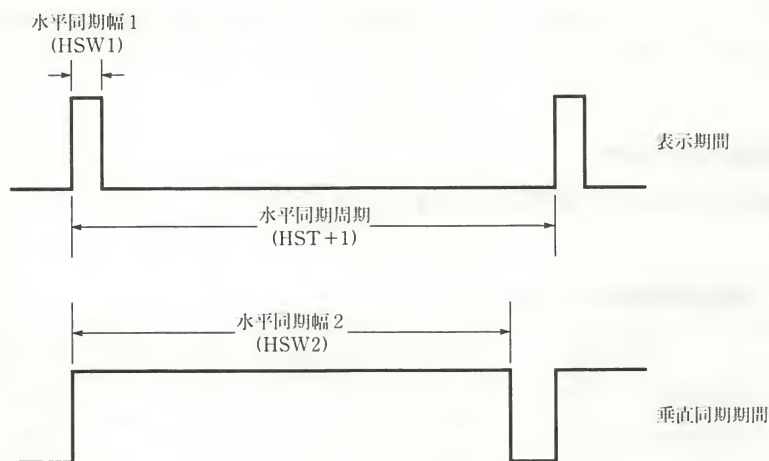
水平同期信号の幅は,輝線の表示期間と垂直同期期間(スポットが画面の最下位から最上位に移動する間の期間)とで別に設定します．

表示期間では，水平同期信号の幅を水平同期幅1レジスタ(HSW1)に設定します．垂直同期期間は，図Ⅰ-4-39のように垂直同期信号があるために，水平同期信号の位相が逆転しており水平同期幅2レジスタ(HSW2)に設定する幅は，水平同期信号の幅以外の部分になります．

設定する値と時間の関係は次の式で表されます．

時間＝(HST＋1)×(メインクロック周期)
時間＝(HSW1)×(メインクロック周期)
時間＝(HSW2)×(メインクロック周期)

なお，メインクロックの周期は，後述のコントロールレジスタ1(表Ⅰ-4-32)のCLKSELに設定した周波数で決まります．

▼図Ⅰ-4-38　水平同期信号関係のレジスタ設定

▼図Ⅰ-4-39　水平同期信号の変化

●垂直同期信号関係のレジスタ

図Ⅰ-4-40に垂直同期信号の周期とレジスタの設定値の関係を示します．

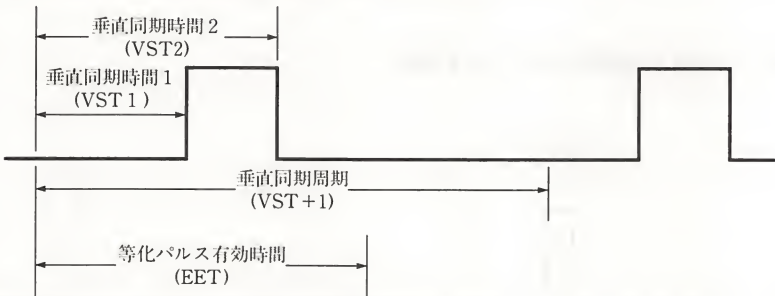
垂直同期信号の周期は，垂直同期周期レジスタ(VST)に設定します．

同期信号のタイミングは，垂直同期時間１レジスタ(VST1)と垂直同期時間２レジスタ(VST2)に設定します．

等化パルス有効時間レジスタ(EET)はインタレース動作を安定させる役割を持っています．

垂直同期信号関係のレジスタでは，設定する値と時間の関係は，次の式で表されます．

時間＝(設定する値)×(HST＋１)×(メインクロック周期)／２

▼図Ⅰ-4-40　垂直同期信号関係のレジスタ設定

●表示区間設定関係のレジスタ

ここでいう表示区間とは，ディスプレイの表示タイミングおよび範囲を決めるものです．

表Ⅰ-4-26に表示区間設定関係のレジスタを示します．

２つの画面レイアに対応して，レジスタも２系統あります．

図Ⅰ-4-41に水平方向の表示区間と水平表示開始位置レジスタ(HDS0, HDS1)，水平表示終了位置レジスタ(HDE0，HDE1)の関係を示します．

この２つのレジスタに設定する値と時間の関係は，次の式で表されます．

時間＝(設定する値)×(メインクロック周期)

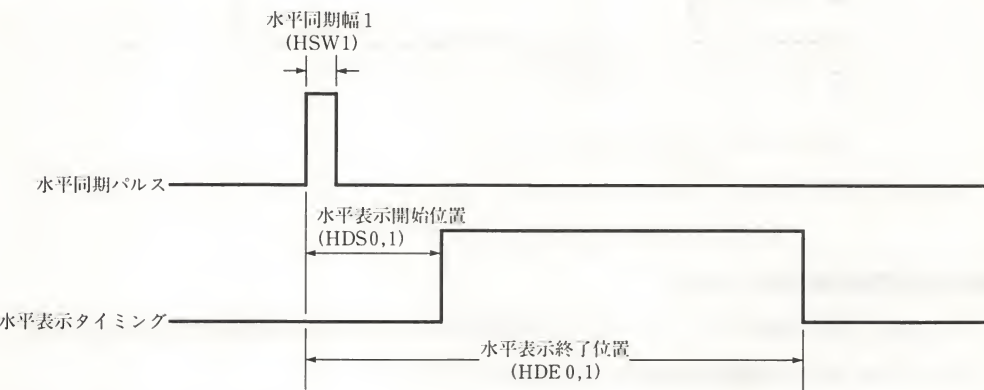
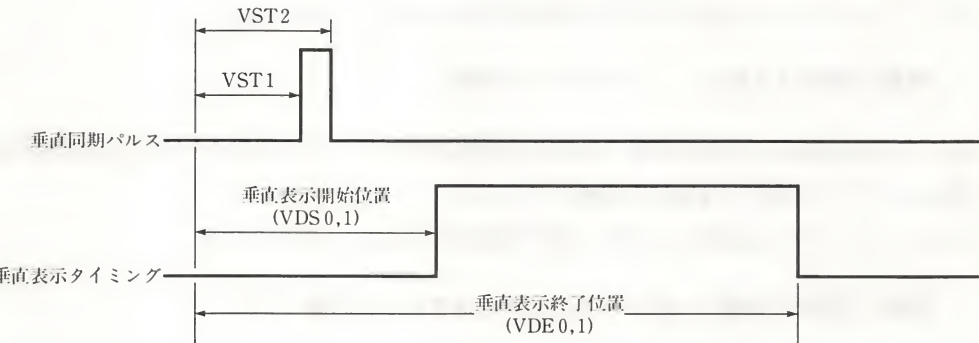
図Ⅰ-4-42に垂直方向の表示区間と垂直表示開始位置レジスタ(VDS0，VDS1)と垂直表示終了位置レジスタ(VDE0，VDE1)の関係を示します．

この２つのレジスタに設定する値と時間の関係は，次の式で表されます．

時間＝(設定する値)×(HST＋１)×(メインクロック周期)／２

▼表 I-4-26　表示区間設定関係のレジスタ

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
09	水平表示開始位置0レジスタ						HDS0										
0A	水平表示終了位置0レジスタ						HDE0										
0B	水平表示開始位置1レジスタ						HDS1										
0C	水平表示終了位置1レジスタ						HDE1										
0D	垂直表示開始位置0レジスタ						VDS0										
0E	垂直表示終了位置0レジスタ						VDE0										
0F	垂直表示開始位置1レジスタ						VDS1										
10	垂直表示終了位置1レジスタ						VDE1										

▼図 I-4-41　水平表示区間関係のレジスタ設定

▼図 I-4-42　垂直表示区間関係のレジスタ設定

●表示アドレス設定関係のレジスタ

表示アドレス設定関係のレジスタ(表 I-4-27)は，画面に表示する仮想画面(VRAM)のアドレスを決めるものです．

２つの画面レイアに対応して，レジスタも２系統あります．

フレーム先頭アドレスレジスタ(FA0，FA1)は，VRAM のどの位置を表示画面の左上角(０，０)にするかを決めるものです．VRAM 領域の相対アドレスで指定します．

また，水平アジャストレジスタ(HAJ)は，シフトレジスタの動作タイミング(１ピクセルを送出するタイミング)をクロックをカウントすることによって得ています．具体値は，標準設定値の表(表 I-4-23)を参考にしてください．

ライン間アドレスオフセットレジスタ(LO0，LO1)は，インタレースモード時に，飛び越し走査のためメモリを飛ばし読みするときの各ライン間の先頭アドレスのずれを決めるものです．CRTC は現在のラインの先頭アドレスにこの値を加算して，次のラインの先頭アドレスを計算します．

また，フィールド間アドレスオフセットレジスタ(FO0，FO1)は，インタレースモード時の０フィールドと１フィールドの先頭アドレスのずれを設定するものです．

なお，フレーム先頭アドレスレジスタの値を増減することにより，画面がスクロールします．レジスタ値の１増減に対してスクロールするピクセル数の関係を表 I-4-28に示します．

▼表 I-4-27　表示アドレス設定関係のレジスタ

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
１１	フレーム先頭アドレス０	FA0															
１２	水平アジャスト０						HAJ0										
１３	フィールド間アドレスオフセット０	FO0															
１４	ライン間アドレスオフセット０	LO0															
１５	フレーム先頭アドレス１	FA1															
１６	水平アジャスト１						HAJ1										
１７	フィールド間アドレスオフセット１	FO1															
１８	ライン間アドレスオフセット１	LO1															

▼表 I-4-28　フレーム先頭アドレスのレジスタ値の１増減に対するスクロールのピクセル数

表示色数	１画面モード	２画面モード
32768色	４ドット	２ドット
256色	８ドット	――
16色	――	８ドット

●外部同期関係のレジスタ

外部同期水平アジャストレジスタ(EHAJ)と外部同期垂直アジャストレジスタ(EVAJ)(表I-4-29)は，スーパーインポーズ時に外部からのビデオ信号とコンピュータ画面の表示のタイミングを調整するためのものです．

外部同期水平アジャストレジスタは，選択しているクロック周波数に合わせて設定します．

また，外部同期垂直アジャストレジスタの設定は，通常，前述の垂直同期時間1レジスタ(VST1)(表I-4-25)の値に1を加えた値を設定します．

▼表I-4-29　外部同期関係のレジスタ

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
１９	外部同期水平アジャスト						EHAJ										
１A	外部同期垂直アジャスト						EVAJ										

EHAJには，クロック周波数に合わせて次のように設定する．

クロック(MHz)	EHAJ
２８．６３６３	００６４H
２５．１７５	００５８H
２４．５４５４	００５６H
２１．０５２５	００４AH

EVAJには，レジスタ番号05のVST1の値に1を加えた値を設定する．

●水平垂直拡大レジスタ

水平垂直拡大レジスタ(表I-4-30)は，画像を任意に拡大するためのレジスタです．拡大率から1を引いた値を設定します．1倍なら0を設定します．

▼表I-4-30　水平垂直拡大レジスタ

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1B	水平垂直拡大レジスタ	ZV1				ZH1				ZV0				ZH0			

ZV1(bit15-12)　：画面レイア1の垂直拡大率．
ZH1(bit11-8)　：画面レイア1の水平拡大率．
ZV0(bit7-4)　：画面レイア0の垂直拡大率．
ZH0(bit3-0)　：画面レイア0の水平拡大率．

●コントロールレジスタ０

コントロールレジスタ０(表Ⅰ-4-31)では，画面レイアごとに設定が可能なものは，２系統のビットが用意されています．

STARTは，画面表示の開始／停止を設定します．

ESYNは，ビデオ信号の同期を外部信号(スーパーインポーズ時)から取るか，コンピュータの信号から取るかを選択するものです．

ESM1, ESM0はスーパーインポーズモードにするか，デジタイズモードにするかを決めるものです．通常は，スーパーインポーズの状態で，ビデオ入力は止めます．

CEN1，CEN0はVRAMのアドレスのカウンタの下位８ビットから上位への桁上がりをするかどうかを決めるもので，０にしてスクロールを行うと円筒スクロールになります．このビットは，640×400ピクセルモードのとき以外は，ディセーブルにします．

CL1，CL0は同時表示色を設定します．設定する値は，画面の数と関係があります．１画面の場合には，CL1はCL0と同じ値を設定します．

▼表Ⅰ-4-31　コントロールレジスタ０(Write)

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1C	コントロールレジスタ０	START	ESYN							ESM1	ESM0	CEN1	CEN0	CL1		CL0	

START(bit15)　：スタート．(このビットのみリードも可)
　０＝ストップ
　１＝スタート

ESYN(bit14)　：外部同期の設定．
　０＝パソコン画面
　１＝外部同期

ESM1(bit7)　：画面レイア１のモード設定．
　０＝スーパーインポーズモード
　１＝デジタイズモード

ESM0(bit6)　：画面レイア０のモード設定．
　０＝スーパーインポーズモード
　１＝デジタイズモード

CEN1(bit5)　：画面レイア１のカウンタキャリーの有無．
　０＝ディセーブル
　１＝イネーブル

CEN0(bit4)　：画面レイア０のカウンタキャリーの有無．
　０＝ディセーブル
　１＝イネーブル

CL1(bit3-2)　：画面レイア１の色数設定．

CL0(bit1-0)　：画面レイア０の色数設定．

CL1，CL0設定値	１画面	２画面
００	――	――
０１	――	32768色
１０	32768色	――
１１	256色	16色

●コントロールレジスタ1

コントロールレジスタ1(表Ⅰ-4-32)のCLKSELは，メインクロックのクロック周波数を選択するものです．

サブキャリアは，色搬送信号として使われます．垂直同期信号期間中にメインクロックを分周して作られ，分周後の2倍の周期を持っています．ここでの設定値は分周比から1を引いた値です．すなわち，0を指定すると1倍になります．

●コントロールレジスタ2

コントロールレジスタ2(表Ⅰ-4-33)は，外部垂直信号分離時のワンショットマルチ回路の動作を規定するものです．

PMには，ワンショット出力のパルス幅をメインクロック数で指定します．

15ビットの目のRETRG(リトリガ)とは，トリガがかかってワンショット動作を行っている最中に，再度トリガがかかった場合，そこを基点として，ワンショット出力を継続することをいいます．

●ダミーレジスタ

ダミーレジスタ(表Ⅰ-4-34)の各ビットは次のような意味があります．

DSPTV1，DSPTV0，DSPTH1，DSPTH0，FIELD，VSYNC，HSYNC，VINは，読み出しのみが可能です．

DSPTV1，DSPTV0，DSPTH1，DSPTH0は各表示期間のとき1になります．

FIELDは，インタレース時にフィールド番号を示します．

VSYNC，HSYNCは，各輝線期間のとき1となります．

VINは，ビデオ入力ありのとき1です．

そのほかのビットは，書き込みのみが可能です．

FR3は，スーパーインポーズ時にハーフトーンにするかどうかを決めます．

FR2は，同期信号生成回路のオン／オフです．

VCRDENは，ビデオカードの動作をするかどうかを決めるものです．

SCENは，色搬送出力の有無を示します．

▼表 I-4-32　コントロールレジスタ 1 (Write)

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1D	コントロールレジスタ 1													SCSEL		CLK SEL	

SCSEL (bit3-2)　：サブキャリアの分周比のセレクト.
0 0 =分周比 2 倍
0 1 =分周比 4 倍
1 0 =分周比 6 倍
1 1 =分周比 8 倍
サブキャリアは，クロックを(SCSEL＋１)の値で分周した周期で交互に反転する.

CLKSEL (bit1-0)　：クロックの選択.
0 0 =28.6363MHz
0 1 =24.5454MHz
1 0 =25.175MHz
1 1 =21.0525MHz

▼表 I-4-33　コントロールレジスタ 2 (Write)

レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1F	コントロールレジスタ 2	RETRG							PM								

RETRG (bit15)　：トリガ設定.
0 =リトリガあり
1 =リトリガなし

PM (bit8-0)　：ワンショット出力パルス幅をCRTCクロック数で指定する．コントロールレジスタ 1 で選択したクロックにより，次の値を指定する.

クロック (MHz)	設定値
28.6363	01CAH
25.175	0192H
24.5454	0188H
21.0525	0150H

▼表Ⅰ-4-34 ダミーレジスタ

		Read												Write			
レジスタ番号	レジスタ名	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1E	ダミーレジスタ	DSPTV1	DSPTV0	DSPTH1	DSPTH0	FIELD	VSYNC	HSYNC	VIN					FR3	FR2	FR1	FR0

DSPTV1(bit15) : 0＝垂直表示期間でない
1＝垂直表示期間

DSPTV0(bit14) : 0＝垂直表示期間でない
1＝垂直表示期間

DSPTH1(bit13) : 0＝水平表示期間でない
1＝水平表示期間

DSPTH0(bit12) : 0＝水平表示期間でない
1＝水平表示期間

FIELD(bit11) : 0＝フィールド0
1＝フィールド1

VSYNC(bit10) : 0＝垂直帰線期間でない
1＝垂直帰線期間

HSYNC(bit9) : 0＝水平帰線期間でない
1＝水平帰線期間

VIN(bit8) : ビデオイン.
0＝なし
1＝あり

FR3(bit3)
HTEN : ハーフトーン.
0＝ハーフトーンにならない
1＝ハーフトーンになる

FR2(bit2)
SGEN : シンクジャネレータイネーブル.
0＝ディセーブル
1＝イネーブル

VCRDEN(bit1)
(FR1) : ビデオカードイネーブル.
0＝ビデオカードは動作しない
1＝ビデオカードは動作する

SCEN(bit0)
(FR0) : サブキャリーイネーブル.
0＝SC信号出力なし
1＝SC信号出力あり

4.8　ビデオ出力制御部と関連レジスタ

画面表示に関わるレジスタには，CRTC の内部レジスタの他にビデオ出力制御部のレジスタがあります．この節では，これらのレジスタについて解説します．

4.8.1　ビデオ出力制御部の関連レジスタ

ビデオ制御部は，VRAM から読み出したデータの重ね合わせ(２つの画面やスプライトなど)，パレット処理，VRAM から並列に読み出された複数ドット分のデータを単一ドットに分解してディスプレイに転送することなどを行っています．

これらの制御に関わるレジスタには次のようなものがあります．

コントロールレジスタ
プライオリティレジスタ
デジタルパレットモディファイフラグ
CRTC 出力コントロールレジスタ
グラフィック VRAM ディスプレイモードレジスタ

このうち，グラフィック VRAM ディスプレイモードレジスタは，FMR-50 互換モードに関係があるので，「4.9　FMR-50 互換の画面表示機能」で解説します．ここでは，それ以外のものについて説明します．

●コントロールレジスタとプライオリティレジスタ

コントロールレジスタ(表 I-4-35)とプライオリティレジスタ(表 I-4-36)への書き込みには，ビデオ出力コントローラ I/O レジスタ(表 I-4-37)を使います．0448H番地にレジスタの番号を設定し，続いて 044AH 番地に値を設定します．

レジスタの番号は２ビットで指定できるようになっていますが，上位ビットは拡張用で，実際には下位の１ビットで，コントロールレジスタとプライオリティレジスタのどちらをアクセスするかを指定します．

コントロールレジスタには，１画面表示の場合と，２画面表示の場合それぞれに対して，ディスプレイの表示を行うかどうかと，色数を指定します．

プライオリティレジスタは，画面レイアの優先順，ビデオ画面の輝度，16色パレットか256色パレットかの選択を行います．

YS は，スーパーインポーズ時の画面切換信号を有効にするかどうかの指定をします．

これらのレジスタは，表 I-4-23の設定値を書き込んでください．

▼表Ⅰ-4-35 コントロールレジスタ

レジスタ番号	レジスタ名	7	6	5	4	3	2	1	0
0 0	コントロールレジスタ	0	0	0	PMODE	CL11	CL10	CL01	CL00

PMODE	CL11	CL10	CL01	CL00	色モード	画　面
0	0	0	0	0	DISPLAY OFF	1画面モード
0	0	1	0	1	DISPLAY OFF	1画面モード
0	1	0	1	0	256色	1画面モード
0	1	1	1	1	32768色	1画面モード
1	*	*	0	0	DISPLAY OFF	(画面レイア0)
1	*	*	0	1	16色	(画面レイア0)
1	*	*	1	0	DISPLAY OFF	(画面レイア0)
1	*	*	1	1	32768色	(画面レイア0)
1	0	0	*	*	DISPLAY OFF	(画面レイア1)
1	0	1	*	*	16色	(画面レイア1)
1	1	0	*	*	DISPLAY OFF	(画面レイア1)
1	1	1	*	*	32768色	(画面レイア1)

*はもう1つの画面レイアの設定値を指定する.

▼表Ⅰ-4-36 プライオリティレジスタ

レジスタ番号	レジスタ名	7	6	5	4	3	2	1	0
0 1	プライオリティレジスタ			PLT0	PLT1	YS	YM		PR1

PLT0-1(bit5-4)　：パレットを選択する.
0 0＝画面レイア0用16色パレット
1 0＝画面レイア1用16色パレット
0 1＝256色パレット
1 1＝256色パレット

YS(bit3)　：
0＝YS有効
1＝YS無効
CRTCが外部同期状態のときに意味がある.
256色外部同期時にはYSは常に有効.

YM(bit2)　：ビデオ画面の輝度を設定.
0＝ビデオ画面高輝度
1＝ビデオ画面低輝度

PR1(bit0)　：画面レイアの優先順を設定.
0＝画面レイア0が前
1＝画面レイア1が前

▼表 I-4-37　ビデオ出力コントローラI/Oレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0448H	アドレスレジスタ	W	0	0	0	0	0	0	RA1	RA0
044AH	データレジスタ	W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0

●デジタルパレットモディファイフラグレジスタ

デジタルパレットモディファイフラグレジスタ(表 I-4-38)の DPMD は，FMR-50 互換のモードにおいて，デジタルパレットレジスタへの書き込みがあったことを知らせるフラグです．

SPD0，PAGE は，スプライトの状態を示すフラグです．

▼表 I-4-38　デジタルパレットモディファイフラグレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
044CH	デジタルパレット モディファイフラグレジスタ	R	DPMD	不　定					SPD0	PAGE

DPMD(bit7)　：デジタルパレットのエミュレーション用フラグ．
　0＝デジタルパレットに書き込みなし
　1＝デジタルパレットに書き込みあり
　このレジスタのリードで0となる．

SPD0(bit1)　：スプライトBUSYフラグ．
　0＝非動作中
　1＝動作中

PAGE(bit0)　：スプライトを展開しているページ．
　0＝ページ0にスプライトを展開，ページ1を表示
　1＝ページ1にスプライトを展開，ページ0を表示

● CRT 出力コントロールレジスタ

CRT 出力コントロールレジスタ(表 I-4-39)は，画面レイア 0 と画面レイア 1 に，表示を行うかどうかを設定します．

GREEN と COLOR のビットのどちらかを 1 にすれば，表示されます．

1 画面表示(VRAM 全体で 512KB 使う場合)では，画面レイア 0 のビットを使用しますが，例外として，32768色の画面モードでは，デジタイズ時には，画面レイア 1 に対しても画面レイア 0 と同じ書き込みをする必要があります．

なお，FM-11 以来の名残りで，GREEN はグリーンディスプレイ，COLOR はカラーディスプレイを示しますが，FMTOWNS では特に区別していません．

▼表I-4-39 CRT出力コントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
							画面レイア0		画面レイア1	
FDA0H	CRT出力コントロールレジスタ	W	0	0	0	0	COLOR	GREEN	COLOR	GREEN

[2画面モード時]

画面レイア0

COLOR	GREEN	意　味
0	0	表示しない
0	1	表示する
1	0	表示する
1	1	表示する

画面レイア1

COLOR	GREEN	意　味
0	0	表示しない
0	1	表示する
1	0	表示する
1	1	表示する

[1画面モード時]

モード	画面レイア0		画面レイア1		表　示
	COLOR	GREEN	COLOR	GREEN	
256色モード	0	0	0	0	表示しない
	0	1	0	0	表示する
	1	0	0	0	表示する
	1	1	0	0	表示する
32768色モード 内部同期，スーパーインポーズ	0	0	0	0	表示しない
	0	1	0	0	表示する
	1	0	0	0	表示する
	1	1	0	0	表示する
32768色モード デジタイズスルー	0	0	0	0	表示しない
	0	1	0	1	表示する
	1	0	1	0	表示する
	1	1	1	1	表示する

これ以外の設定をした場合表示の保証はされない．

4.9　FMR-50互換の画面表示機能

FMTOWNS は，FMR-50 のアプリケーションが使用できるように設計されており，画面表示についても FMR-50 と見かけ上，同様の表示ができます．FMR-50 互換の画面表示は，ハードウェアと BIOS の対応によって実現されていますが，この節では，ハードウェアレベルでの互換の仕組みについて説明します．

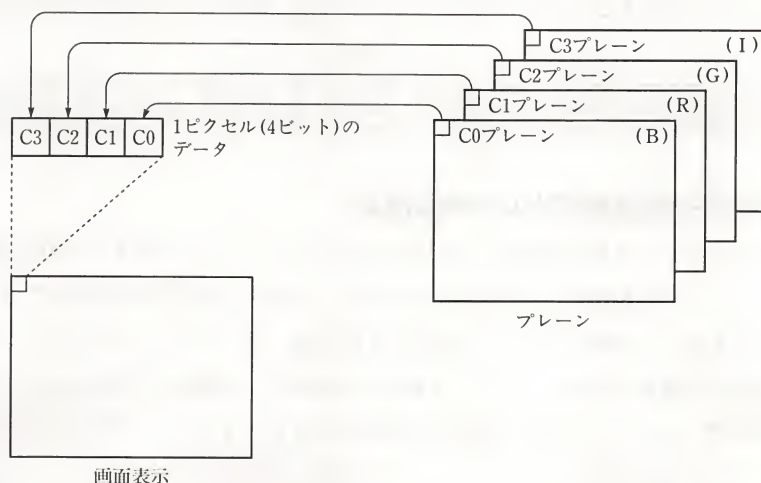
4.9.1　FMR-50の画面表示

FMR-50 には，640×400ピクセルのグラフィック画面があり，パレットを使って，4096色中16色，16色中8色の表示が可能です．また，最大80桁×25行のテキスト画面があり，16色の文字表示が可能です．

グラフィック画面の16色表示は，4ビットのデータの ON，OFF の組み合わせで行っています．しかし，ピクセル単位に色ビットがまとまって配置されているわけではありません．ピクセルの第3ビット(C3)，第2ビット(C2)，第1ビット(C1)，第0ビット(C0)のデータが，それぞれ別個に VRAM に格納されています．そして個別のビット単位のデータの集まりはプレーンと呼ばれています．VRAM の読み書きはこのプレーンを単位にして行われており，このような VRAM をプレーンアクセス VRAM といいます．プレーンの概念を図Ⅰ-4-43に示します．また，16色から選ぶ8色は，デジタルパレットに格納するようになっています．

FMR-50 では，テキストの表示は，テキスト VRAM を使用しており，文字を表示するには，キャラクタコードを VRAM に書き込みます．

FMTOWNS では，グラフィック画面については，FMR-50 と同様にプレーンアクセスが可能になっています．テキスト表示は，グラフィック画面を使用します．

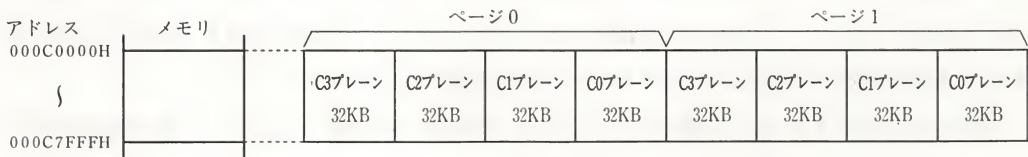
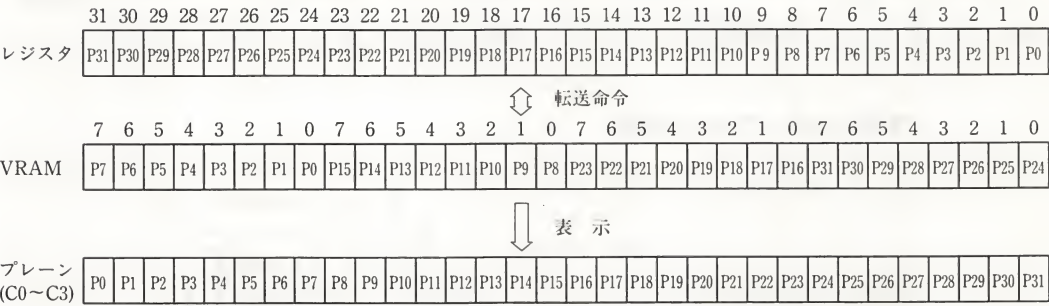
▼図Ⅰ-4-43　プレーンの概念

4.9.2　FMR-50互換の VRAM のプレーンアクセス

FMTOWNS で, FMR-50 互換の画面表示を行う際には, まず, FMR-50 互換の画面モード(画面レイア 0 は画面モード 1, 画面レイア 1 は画面モード 4)にします. 画面レイア 0 をグラフィック画面用に, 画面レイア 1 をテキスト表示用に使用します.

FMTOWNS の VRAM は, 通常はビットマップ形式ですが, ページ 0 とページ 1 は, プレーンアクセスが可能です.

VRAM の 1 ページは 128KB ありますから, 各プレーンの大きさは 32KB であり, プレーンの読み書きには, C0000H からの 32KB のメモリを窓口として使用します(図 I-4-44). 表示ページの指定は, グラフィック VRAM ディスプレイモードレジスタ(表 I-4-41)で, 書き込みページの指定は, グラフィック VRAM ページセレクトレジスタ(表 I-4-44)で, どのプレーンを読み書きするかは, グラフィック VRAM 更新モードレジスタ(表 I-4-43)で設定します. なお, CPU のレジスタの並び順と, 各プレーンへ転送されたときの並び順は図 I-4-45のようになります.

なお, FMR-50 互換の 8 色表示では, C 3 プレーンは使われません.

▼図 I-4-44　プレーンアクセス時の VRAM のアドレス

▼図 I-4-45　プレーンデータ転送時のレジスタ, VRAM の内容

4.9.3　FMR-50互換のパレットの指定

16色中の 8 色のパレット色の設定は, デジタルパレットレジスタ(表 I-4-40)を使用します. しかし, このパレットは FMR-50 互換のためのダミーであり, 書き込みを行っても色変換には何の影響もありません. 実際にパレットを設定するには, ソフトウェアで, デジタルパレットレジスタを参照して得た値をもとにして, FMR-50 のパレット設定と同等になるように色の配置をし, アナログパレットレジスタに書き込まなければなりません.

なお，デジタルパレットレジスタへの書き込みがあったかどうかは，デジタルパレットモディファイフラグのDPMDを参照して調べることができます.

4096色中16色表示の場合は，アナログパレットレジスタを使用します.

▼表Ｉ-4-40　デジタルパレットレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FD98H	パレットデータ 0	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD99H	パレットデータ 1	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD9AH	パレットデータ 2	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD9BH	パレットデータ 3	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD9CH	パレットデータ 4	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD9DH	パレットデータ 5	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD9EH	パレットデータ 6	R	不定				C3	C2	C1	C0
		W	0	0	0	0				
FD9FH	パレットデータ 7	R	不定				C3	C2	C1	C0
		W	0	0	0	0				

このレジスタはFMR-50互換用のダミーレジスタで，単に読み/書きできるだけなので，使用するときはソフトウェアで本来の機能を代行する必要がある.

4.9.4　FMR-50互換の文字表示

FMR-50は，テキストVRAMを使用していますが，FMTOWNSでは，VRAMの画面レイア1を画面モード4の設定で使います. BIOSを使わずに文字を表示するには，キャラクタジェネレータを呼び出して，文字のフォントに対応するドットを画面レイア1に書き込みます.

4.9.5　FMR-50互換モードに関連するレジスタ

FMR-50互換モードに関連するレジスタを表Ｉ-4-41，表Ｉ-4-42，表Ｉ-4-43，表Ｉ-4-44，表Ｉ-4-45，表Ｉ-4-46に示します.

●グラフィックVRAMディスプレイモードレジスタ

PS2には，FMR-50互換モード時に，グラフィック画面にVRAMのページ0とページ1のどちらを表示するかを設定します. スプライト静止表示時には，0にしておかなければなりません.

RAM1-4には各プレーンの表示の有無を設定します.

▼表Ⅰ-4-41　グラフィックVRAMディスプレイモードレジスタ

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="3">000C
FF82H</td><td rowspan="3">グラフィックVRAM
ディスプレイモードレジスタ</td><td>R</td><td colspan="8">不　　定</td></tr>
<tr><td rowspan="2">W</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">RAM4</td><td colspan="2">PAGE SELECT</td><td colspan="3">RAM SELECT BIT</td></tr>
<tr><td>PS2</td><td>0</td><td>RAM3</td><td>RAM2</td><td>RAM1</td></tr>
</table>

FMR-50互換レジスタ

RAM4(bit5)　：C3プレーンの表示.
　0＝表示しない
　1＝表示する

PS2(bit4)　：表示ページの指定.
　0＝ページ0
　1＝ページ1

RAM3(bit2)　：C2プレーンの表示.
　0＝表示しない
　1＝表示する

RAM2(bit1)　：C1プレーンの表示.
　0＝表示しない
　1＝表示する

RAM1(bit0)　：C0プレーンの表示.
　0＝表示しない
　1＝表示する

● MIXレジスタ

テキスト画面の1行当たりの表示文字指定や，カーソルポジションの最下位ビットを指定します．このレジスタは，ダミーですから，ソフトウェアによるエミュレーションによって，ページ2あるいは，ページ3への文字の出力を行う必要があります．

▼表Ⅰ-4-42　MIXレジスタ

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">000C
FF80H</td><td rowspan="2">MIXレジスタ</td><td>R</td><td colspan="2">不　定</td><td rowspan="2">CURSOR
LSB</td><td>不定</td><td rowspan="2">WIDTH</td><td colspan="3">不　　定</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

FMR-50互換用ダミーレジスタ

CURSORLSB(bit5)　：カーソルポジションの最下位ビット.

WIDTH(bit3)　：1行の表示文字数.
　0＝40文字
　1＝80文字

●グラフィック VRAM 更新モードレジスタ

このレジスタは，CPU が VRAM の読み書きをする際にプレーンの指定を行うものです．書き込み時に複数のプレーンを指定すると同時に書き込みが行われます．

▼表 I-4-43　グラフィックVRAM更新モードレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF81H	グラフィックVRAM 更新モードレジスタ	R	READOUT CNTRL		不　定		RAM SELECT BIT			
			RC2	RC1			RAM4	RAM3	RAM2	RAM1
		W	READOUT CNTRL		0	0	RAM SELECT BIT			
			RC2	RC1			RAM4	RAM3	RAM2	RAM1

FMR-50互換用レジスタ

RC2-1 (bit7-6)　：VRAMを読み出すときの対象プレーン．
0 0 =C0プレーンの読み出し
0 1 =C1プレーンの読み出し
1 0 =C2プレーンの読み出し
1 1 =C3プレーンの読み出し

RAM4 (bit3)　：C3プレーンへの書き込み．
0 =書き込まない
1 =書き込む

RAM3 (bit2)　：C2プレーンへの書き込み．
0 =書き込まない
1 =書き込む

RAM2 (bit1)　：C1プレーンへの書き込み．
0 =書き込まない
1 =書き込む

RAM1 (bit0)　：C0プレーンへの書き込み．
0 =書き込まない
1 =書き込む

書き込み時には，同時に複数プレーンの書き込みが可能．

●グラフィック VRAM ページセレクトレジスタ

このレジスタは，CPU が VRAM の書き込みをする際にページの指定をするものです．

▼表 I-4-44　グラフィックVRAMページセレクトレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF83H	グラフィックVRAM ページセレクトレジスタ	R	不　定			PAGE SELECT		不　定		
						PS2	0			
		W	0	0	0	PAGE SELECT		0	0	0
						PS2	0			

FMR-50互換用レジスタ

PS2 (bit4)　：VRAMの読み書きを行うページの選択．
0 =ページ 0
1 =ページ 1

● SUB ステータスレジスタ

CRT 出力コントロールレジスタと同じ I/O アドレスにあり，読み出し時に水平，垂直同期信号の ON/OFF を参照できます．

▼表 I-4-45　SUBステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FDA0H	SUBステータスレジスタ	R	不定						HSYNC	VSYNC

FMR-50互換用レジスタ

HSYNC(bit1)　：水平帰線期間中であることを示す．
　0＝水平帰線期間でない
　1＝水平帰線期間

VSYNC(bit0)　：垂直帰線期間中であることを示す．
　0＝垂直帰線期間でない
　1＝垂直帰線期間

● STATUS レジスタ

このレジスタは，主として表示期間であるかどうかを確認するのに用いられます．垂直同期期間のときは，表示画面に直接アクセスしても画面表示がちらつくことはありません．

▼表 I-4-46　STATUSレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF86H	STATUSレジスタ	R	HSYNC	不定		1	不定	VSYNC	不定	

メモリマップドI/OでのFMR-50互換用レジスタ

HSYNC(bit7)　：水平帰線期間中であることを示す．
　0＝水平帰線期間でない
　1＝水平帰線期間

VSYNC(bit2)　：垂直帰線期間中であることを示す．
　0＝垂直帰線期間でない
　1＝垂直帰線期間

4.10　表示システムのメモリマップとI/Oアドレス

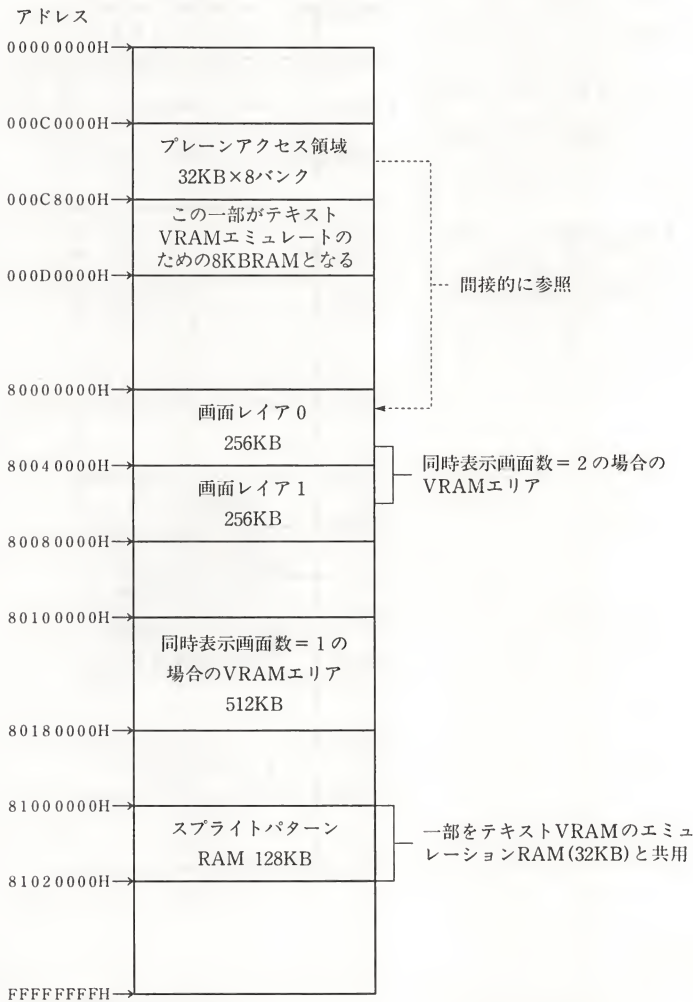
この節では，表示システムに関わるメモリとI/Oアドレスを示します．

4.10.1　表示システムのメモリマップ

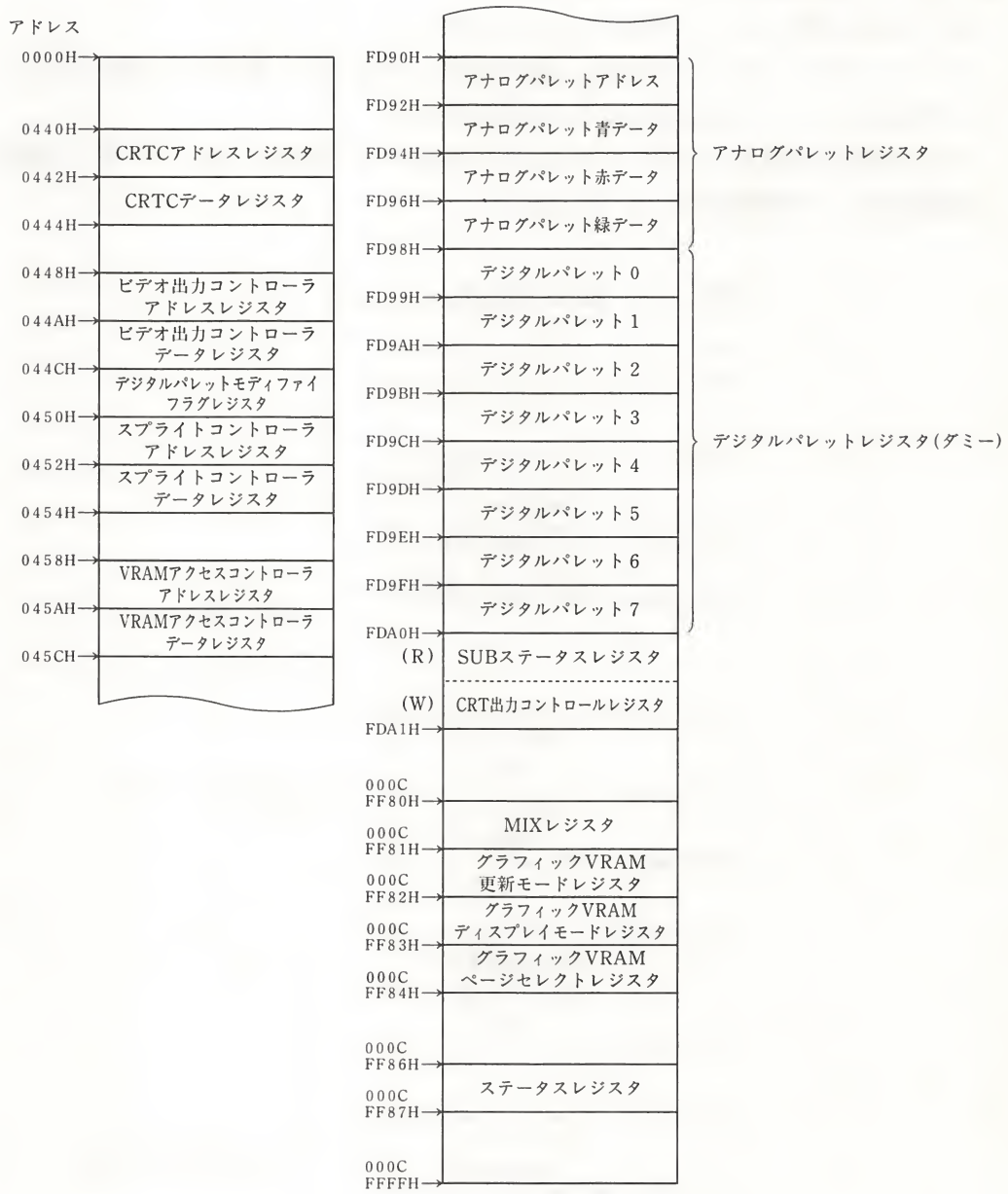
図I-4-46に，表示システムで利用するメモリの割り当てを示します．

スプライトパターンメモリ(128KB)の一部(テキストVRAM4KB，漢字VRAM4KB)はテキストVRAMエミュレート用のRAMとして使われます．

また，FMR-50互換モード時には，グラフィック側のメモリ領域は，プレーンアクセス領域への読み書きにより間接的にアクセスされます．

▼図I-4-46　表示システムのメモリマップ

4.10.2 表示システムのI/Oアドレスマップ

図I-4-47に，表示システムに関係するI/Oアドレスマップを示します.

▼図I-4-47　表示システムのI/Oアドレスマップ

4.11　ビデオカード

ビデオカードは，ビデオコンバート，ビデオデジタイズ，スーパーインポーズを行うためのハードウェアです．ビデオカードコネクタに挿入して使用します．この節では，ビデオカードの仕組みと機能について説明します．

4.11.1　ビデオカードのハードウェア仕様

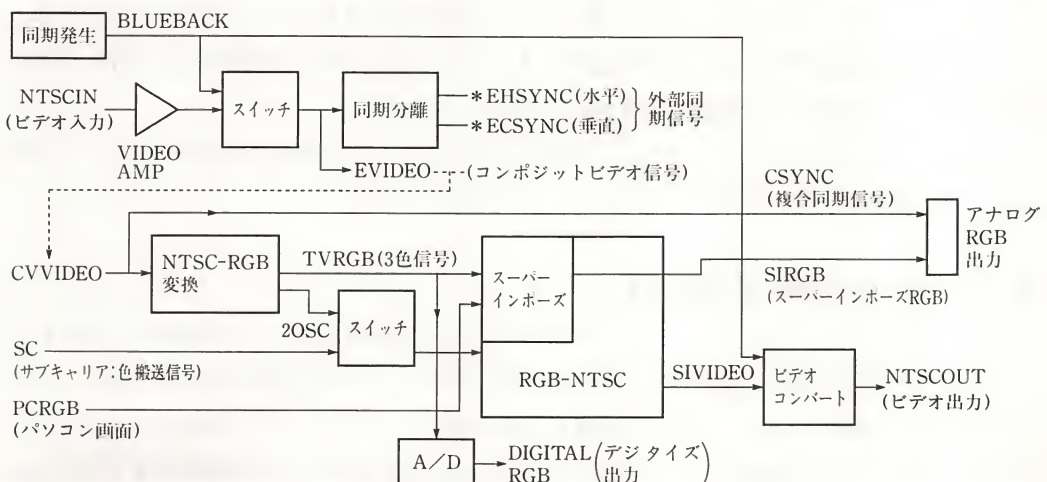
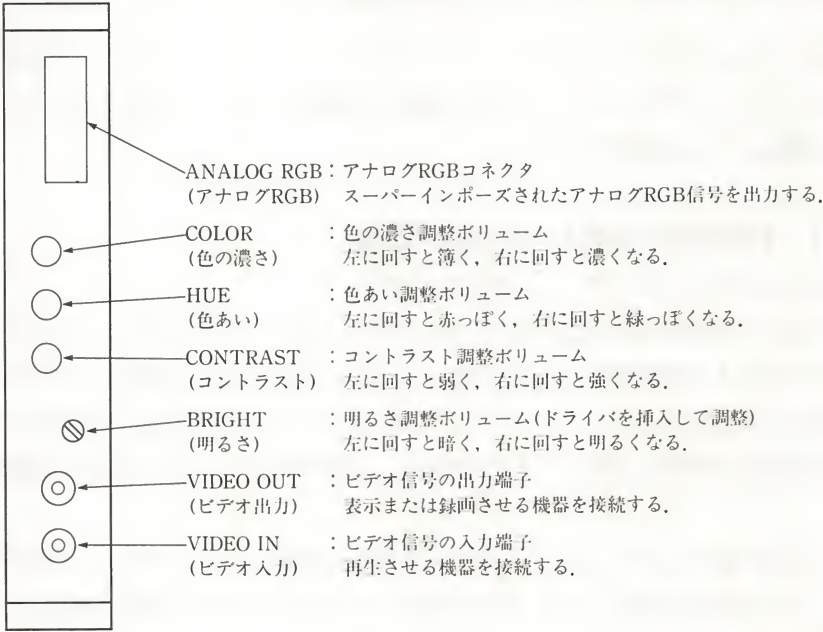
図I-4-48にビデオカードのブロック図を示します．

ビデオ入力は，外部からの入力用に使うもので，ここからの入力に対してスーパーインポーズやビデオデジタイズ処理を行うことができます．ビデオ出力は，FMTOWNSのアナログRGB出力信号をビデオ信号にコンバートしたものです．ビデオ入力と出力は，一般のビデオコンポジット信号(NTSC信号)に対応したものですから，家庭用の種々のビデオ機器の接続が可能です．

アナログRGB端子は，スーパーインポーズを行った画面をディスプレイに出力させるためのもので，FMTOWNS本体のアナログRGB端子と一部を除きほぼ同様の規格のものです．スーパーインポーズ時以外でも使用することはできますが，水平周波数が，15.73KHzの場合だけしか周波数特性が保証されないので，その他の場合には，ディスプレイは本体のRGBコネクタに接続したほうがよい結果が得られます．

また，ビデオカード背面パネル(図I-4-49)にあるつまみにより，表I-4-47に示すような調整を行うことができます．

アナログRGBコネクタ，ビデオカードコネクタのピンの配置とその意味などについては，巻末の付録を参照してください．

▼図I-4-48　ビデオカードのブロック図

▼図I-4-49 ビデオカード背面パネルの端子等の配置

▼表I-4-47 ビデオカードの外部調整

色あいVR	ビデオ出力，デジタイズの色あい調整
色の濃さVR	ビデオ出力，デジタイズの色の濃さ調整
明るさVR	デジタイズの明るさ調整
コントラストVR	デジタイズのコントラスト調整

4.11.2 ビデオコンバート

ビデオコンバートは，アナログRGB信号をビデオ信号化するもので，この機能を使用するとFMTOWNSの画面出力をビデオ信号に変換できます．ただし，水平周波数が15.73KHz以外の場合には，ビデオ信号への変換は保証されません．

表示のモードにより，表I-4-48のような出力となります．この表中のブルーバックビデオ出力は，ビデオコンバートが不完全であることを示すものです．

4.11.3 スーパーインポーズ

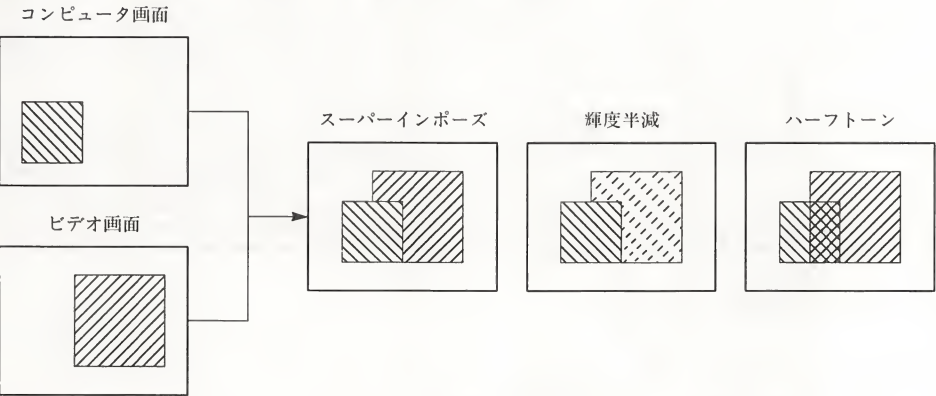
外部からのビデオ入力信号は，アナログRGB信号に変換されたあと，本体側から入力されたコンピュータ画面と重ね合わせされ，アナログRGB出力とビデオ出力から出力されます．スーパーインポーズ時には，ビデオ画面の輝度を半減させたり，コンピュータ画面をハーフトーンにして，コンピュータ画面とビデオ画面をミックスした表示をすることができます(図I-4-50)．

▼表 I-4-48　ビデオコンバート時の同期信号などの対応関係

1	低解像度（15・75KHz）	NTSC準拠モード（クロック 28.6363MHz）	PC画面	ソースなし	FMTOWNS本体(CSYNC)に同期
2				ソースあり	
3			SI画面	ソースなし	
4				ソースあり	ビデオソースに同期
5		NTSC準拠以外のモード	PC画面	ソースなし	FMTOWNS本体(CSYNC)に同期
6				ソースあり	
7			SI画面	ソースなし	
8				ソースあり	ビデオソースに同期
9	中，高解像度(24KHz/31KHz)		PC画面	ソースなし	ブルーバックビデオ出力
10				ソースあり	

PC画面　：パソコン画面のみ
SI画面　：スーパーインポーズ画面(デジタイズ表示を含む)
ソース　：ビデオソース信号

５～７項のときは，色同期信号とビデオ出力が非同期となり色流れ，色にじみが発生する

▼図 I-4-50　スーパーインポーズの形態

4.11.4 ビデオデジタイズ

外部からのビデオ入力信号は，アナログ RGB 信号に変換された上で，A/D コンバータでデジタル化され，本体側の VRAM に書き込まれます．取り込みは32768色(RGB 各５ビット)で行われます．CRTC のコントロールレジスタの ESM0，ESM1 でデジタイズ状態にする画面レイアの指定を行います．

4.11.5 スーパーインポーズとビデオデジタイズ時のレジスタ設定

スーパーインポーズとビデオデジタイズ時のレジスタ設定をするには，画面モードがスーパーインポーズやビデオデジタイズが可能なモードになるように，表Ⅰ-4-23に合わせて，各レジスタの設定を行います．スーパーインポーズの場合には，画面モードを9，11，14，18，ビデオデジタイズの場合には，画面モードを9，11，18のどれかにします．

このとき，CTRCの内部レジスタとビデオ制御部のレジスタの必要なビットを表Ⅰ-4-49に合わせて変更する必要があります．

▼表Ⅰ-4-49　レジスタの設定値と出力信号の関係

レジスタ								本体		ビデオカード					
CRTC内部レジスタ						＊		アナログRGB出力		アナログRGB出力			ビデオ出力		
ESYN	ESM0	ESM1	VCR DEN	FR2	FR3	Ys	Ym	画面レイア0	画面レイア1	画面レイア0	画面レイア1	ビデオレイア	画面レイア0	画面レイア1	ビデオレイア
0	0	0	0	0		1		ＰＣ内	ＰＣ内	ＰＣ内	ＰＣ内	未表示	未表示	未表示	ブルーバック
0	0	0	1	0		1		ＰＣ内	ＰＣ内	ＰＣ内	ＰＣ内	未表示	ＰＣ内	ＰＣ内	未表示
1	0	0	1	0,1		1		ＰＣ外	ＰＣ外	ＰＣ外	ＰＣ外	未表示	ＰＣ外	ＰＣ外	未表示
1	0	1	1	0		1		ＰＣ外	デジタイズ	ＰＣ外	デジタイズ	未表示	ＰＣ外	デジタイズ	未表示
1	1	0	1	0		1		デジタイズ	ＰＣ外	デジタイズ	ＰＣ外	未表示	デジタイズ	ＰＣ外	未表示
1	1	1	1	0		1		デジタイズ	デジタイズ	デジタイズ	デジタイズ	未表示	デジタイズ	デジタイズ	未表示
1	0	0	1	0	0,1	0	0,1	ＰＣ外	ＰＣ外	スーパー	スーパー	ビデオ	スーパー	スーパー	ビデオ
1	0	0	1	1	0,1	0	0,1	ＰＣ外	ＰＣ外	スーパー	スーパー	ブルーバック	スーパー	スーパー	ブルーバック
1	0	1	1	0	0,1	0	0,1	ＰＣ外	デジタイズ	スーパー	デジタイズ	ビデオ	スーパー	デジタイズ	ビデオ
1	1	0	1	0	0,1	0	0,1	デジタイズ	ＰＣ外	デジタイズ	スーパー	ビデオ	デジタイズ	スーパー	ビデオ
1	1	1	1	0	0,1	0	0,1	デジタイズ	デジタイズ	デジタイズ	デジタイズ	ビデオ	デジタイズ	デジタイズ	ビデオ

ＰＣ内　：内部同期のパソコン表示
ＰＣ外　：外部同期のパソコン表示
スーパー　：スーパーインポーズ表示
デジタイズ　：デジタイズスルー表示
ブルーバック：シンクジェネレータの出力(青１色)
ビデオ　：ビデオ入力端子から入力されるビデオ信号

注)・＊はビデオ出力制御のプライオリティレジスタ．
・表以外の設定を行わないこと．
・FR2＝０，ESYN＝１のとき，ビデオ入力がない場合，内部同期のパソコン表示になる．
・Ym＝１のとき，ビデオレイアは低輝度になる．
・FR3＝１のとき，画面レイア０，画面レイア１の表示が半透明になり，ビデオレイアが表示される．
・フレームバッファの取り扱いについて
ESYN　０
FR2　　１
} のとき，Ys＝０と同価にする．

なお，外部同期時には，次のような注意が必要です．

32768色１画面のモードの場合

スーパーインポーズ時：画面レイア１をDISPLAY OFFにしておく(ONの場合はスーパーインポーズしない)．

ビデオデジタイズ時：画面レイア１をDISPLAY ONにしておく(OFFの場合はデジタイズ表示ができない)．

256色１画面モードの場合

スーパーインポーズ時：YSの有効／無効の切り替えはできない．常にYSは有効．

第 5 章

オーディオシステム

FMTOWNSの音楽機能は，従来のパソコンに比べ，飛躍的にパワーアップされています．

CDプレーヤとしても使用できるCR-ROMドライブを標準で装備しているほか，ステレオ出力が可能なFM音源とPCM音源が用意されており，多彩な音楽表現が可能です．また，内蔵マイクやマイク端子，オーディ入出力端子，ヘッドホン端子を装備しており，他のオーディオ機器を接続することができます．

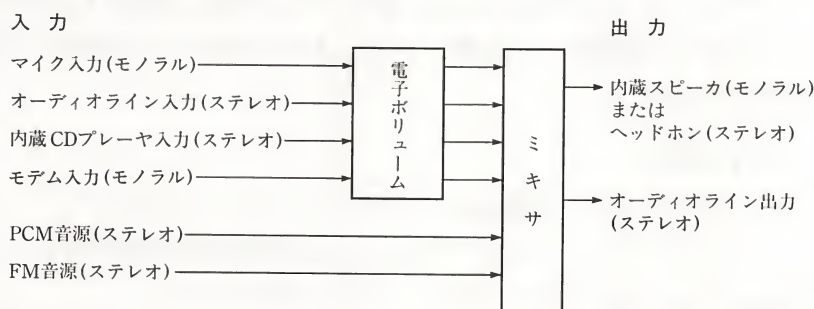
この章では，FMTOWNSの優れたオーディオシステムのハードウェアの仕組みについて解説します．

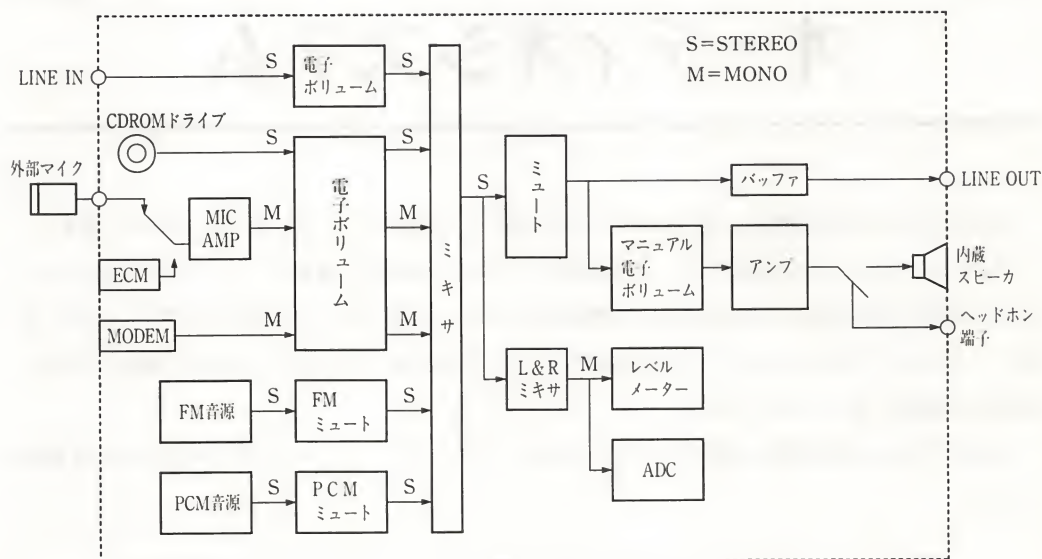
5.1 オーディオシステムの概要

この節では，FMTOWNSのオーディオシステムの構成，入出力信号などの全体的な解説を行います．個々の部分の詳細な説明は，次節以降で行います．

5.1.1 オーディオシステムの構成

図I-5-1に，FMTOWNSのオーディオシステムのすべての入力信号と出力信号を示します．

▼図I-5-1 FMTOWNSで扱うオーディオ信号

FMTOWNSは，多種類の入力信号を内部の電子ボリュームとミキサでミキシング(混合)して出力する，という考えで設計されています．いわば，「コンピュータ制御のミキサ」を内蔵しているといえます．オーディオシステムのブロック図を，図Ⅰ-5-2に示します．

▼図Ⅰ-5-2 オーディオシステムブロック図

以下に，各部の概要を記します．

●マイク入力

本体前面の左下に，エレクトレットコンデンサマイクを内蔵しています．また，外部マイク端子に，外付けのマイクを接続して使用できます．この場合には，内蔵マイクは切り離されて使用できなくなります．

マイク入力信号は，モノラルで処理されます．外部マイク端子はモノラルミニジャック用で，それ以外のマイクを使う場合には，市販の変換プラグを利用します．

マイク入力は，マイクロホンの出力レベルが低いので，他の入力とバランスを取るために専用のアンプで増幅されてから，電子ボリュームに送られます．

PCMサンプリングで高音質に録音するには，録音レベルを最適に調整する必要があります．外部マイクはできるだけ高感度のものが適しており，カタログ感度−55dB(0db=1V/Pa)以上のマイクが推奨されます．

●オーディオライン入力

オーディオライン入力には，一般的なオーディオシステム(ステレオアンプ，テープデッキ，CDプレーヤ，テレビ，ビデオ，ラジオ，ラジカセなど)の音声出力が接続できます．オーディオライン入力は，左右2系統が独立しており，ステレオで処理されます．

オーディオラインからの入力信号は，そのまま電子ボリュームへ送られます．

●内蔵CD-ROMドライブ入力

内蔵されているCD-ROMドライブは，オーディオCDのプレーヤとして利用することができます．CD-ROMドライブからの信号は，左右2系統が独立しており，ステレオで処理されます．CD-ROMドライブからの入力信号は，そのまま電子ボリュームへ送られます．

●PCM音源

PCM音源の再生時には，8チャネルの音を同時に再生することができます．また，各チャネルごとに左右への振り分けを行い，ステレオで出力することが可能です．PCM音源の出力はレベル調整が可能なため，電子ボリュームを通さず，直接ミキサへ送られます．

●FM音源

6チャネルの音を同時に発生させることができ，各チャネルごとにステレオ(左右の一方または，両方)となります．FM音源の出力も，レベル調節が可能なため，電子ボリュームを通さず，ミキサへ送られます．なお，従来FMシリーズの下位機種にあったPSG音源(旧FM音源LSIに内蔵)は削除されています．

●FM／PCMミュート回路

FM音源，PCM音源の出力をミュートする回路です．ミュートの作動／解除は，FM・PCMミュートレジスタで設定します．

●電子ボリューム

入力信号の音量は，電子ボリュームを使って調節することができます．電子ボリューム制御用のレジスタをCPUからアクセスすることによって調節します．

●ミュート回路

ミキサ出力をショートし，出力端子の信号を止める回路です．リセット直後は強制的に作動状態(切り離されている)にセットされています．ミュートの作動／解除は，オーディオレジスタに設定します．

●バッファ

バッファは，出力端子の直前に置かれていて，出力端子に接続されている装置の影響を緩和し，レベルの低下や音質の劣化を防ぎます．出力インピーダンスはおよそ1KΩに設定されています．

●内蔵スピーカ

ミュート回路を経た信号は，スピーカアンプで増幅されて内蔵スピーカに出力されます．このとき，アンプの左右の出力を均等に混合したモノラル信号となります．音量は手動のボリュームで調節できます．音量レベルはステータスインジケータ(LED)で表示されます．

●ヘッドホン出力

ヘッドホン出力端子にヘッドホンを接続すると，ミュート回路を経た出力信号をステレオで聞くことができます．なお，ヘッドホンのプラグを挿入すると，スピーカの音は聴こえなくなります．音量の調節とLEDの点灯に関しては内蔵スピーカと同様です．

●オーディオライン出力

左右2系統(ステレオ)の出力端子があります．一般的なオーディオシステムの音声入力に接続することができます．

5.2　電子ボリュームと減衰量設定について

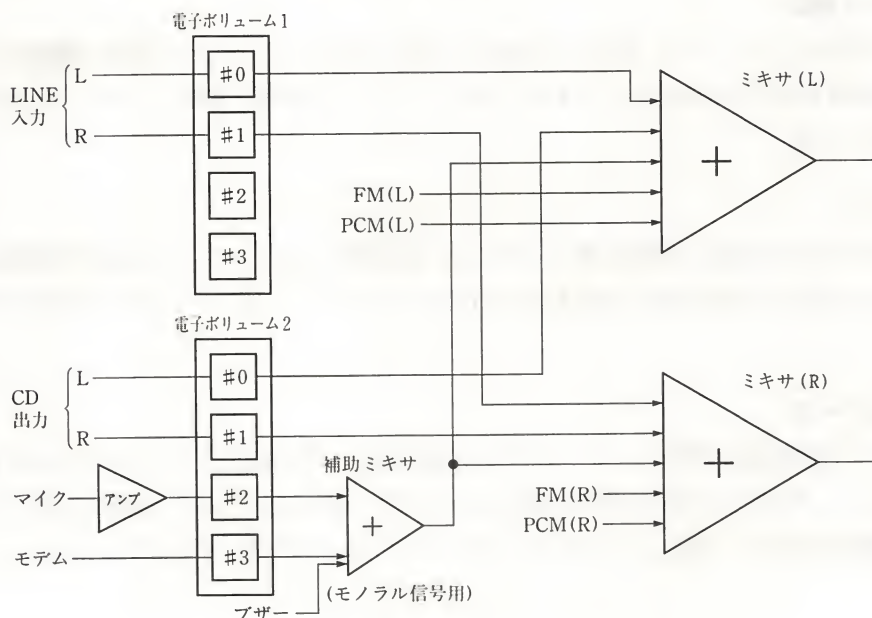
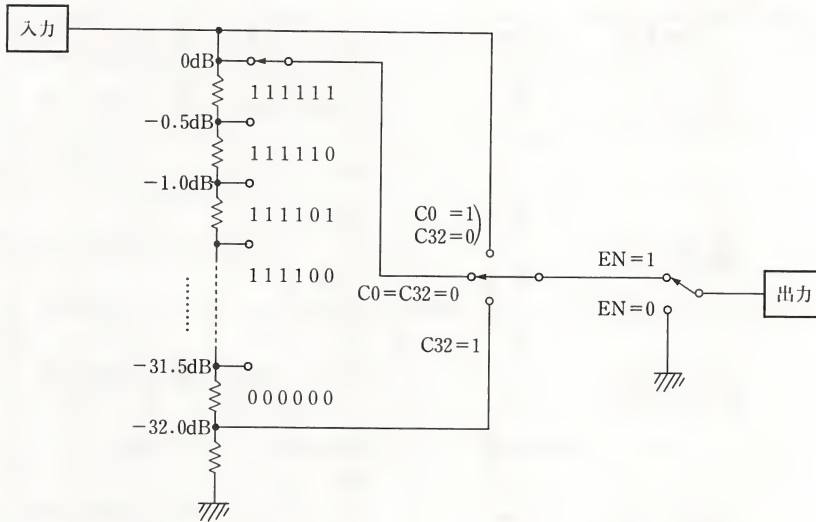
FMTOWNSでは，各オーディオ入力信号の音量調節に電子ボリュームを使用しています．この節では，電子ボリュームの構造と減衰量の設定の仕組みについて説明します．

5.2.1　電子ボリュームとチャネル

電子ボリュームには富士通のMB87078が2個使われています．このICには4個の電子ボリュームが内蔵されており，合計8つの信号の音量調節が可能です．

電子ボリュームの割り当ては，図Ⅰ-5-3のようになっており，実際に使用されているのは6チャネルです．

各チャネルの電子ボリュームは図Ⅰ-5-4に示す構造になっています．

▼図Ⅰ-5-3　電子ボリュームのチャネル割り当て

▼図Ｉ-5-4　電子ボリュームの各チャネルの構造

5.2.2 電子ボリュームレジスタによる減衰量の制御

減衰量は，CPU から電子ボリュームレジスタ（表Ｉ-5-1）に書き込みを行うことによって制御します．

電子ボリュームレジスタは，電子ボリュームの数に対応して２系統用意されています．それぞれ，DATA レジスタと COM レジスタがあります．

音量を大きく変えるには，COM レジスタを使います．各チャネルごとに EN＝０とすると−∞dB（消音），C32＝１では−32.0dB，C０＝１かつ C32＝０では 0dB といったように音量を変えることができます．チャネルは CH0，CH1 で設定します．

音量を微妙に変えるには，DATA レジスタの下位６ビットを使います．各チャネルの減衰量の大きさを64段階に設定できます．

各チャネルの音の大きさは，表Ｉ-5-2のように６ビット（０から63）の値によって設定されており，６ビットのデータ値が０のとき−31.5，63のとき 0dB となります．

減衰量と実際の音の大きさの比率（伝送比率）の対応関係は表Ｉ-5-3のようになります．

表Ｉ-5-3の中で，0dB は１倍，−3dB は約1/$\sqrt{2}$倍，−6dB は約1/2倍，−20dB は1/10倍といった関係を覚えておくと，その他の減衰率のときの信号の伝送比率を計算できるので便利です．例えば，−26dB の場合は次のようになります．

−26dB＝−20dB＋（−６dB）…………1/10×1/2＝1/20

このように，減衰量（dbB）の足し算は，伝送比率の計算では掛け算になるので，−26dB では伝送比率は1/20と計算できます．

▼表I-5-1　電子ボリュームレジスタ

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">04E0H</td><td rowspan="2">ボリューム1
DATAレジスタ</td><td>R</td><td colspan="2">不　定</td><td rowspan="2">D5</td><td rowspan="2">D4</td><td rowspan="2">D3</td><td rowspan="2">D2</td><td rowspan="2">D1</td><td rowspan="2">D0</td></tr>
<tr><td>W</td><td>0</td><td>0</td></tr>
<tr><td rowspan="2">04E1H</td><td rowspan="2">ボリューム1
COMレジスタ</td><td>R</td><td colspan="3">不　定</td><td rowspan="2">C32</td><td rowspan="2">C0</td><td rowspan="2">EN</td><td rowspan="2">CH1</td><td rowspan="2">CH0</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td rowspan="2">04E2H</td><td rowspan="2">ボリューム2
DATAレジスタ</td><td>R</td><td colspan="2">不　定</td><td rowspan="2">D5</td><td rowspan="2">D4</td><td rowspan="2">D3</td><td rowspan="2">D2</td><td rowspan="2">D1</td><td rowspan="2">D0</td></tr>
<tr><td>W</td><td>0</td><td>0</td></tr>
<tr><td rowspan="2">04E3H</td><td rowspan="2">ボリューム2
COMレジスタ</td><td>R</td><td colspan="3">不　定</td><td rowspan="2">C32</td><td rowspan="2">C0</td><td rowspan="2">EN</td><td rowspan="2">CH1</td><td rowspan="2">CH0</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td></tr>
</table>

EN　：電子ボリュームの出力制御信号　0のとき－∞のレベル．
C32　：電子ボリュームの出力制御信号　1のとき－32dBのレベル．
C0　：電子ボリュームの出力制御信号　1のとき0dBのレベル．
D5-0　：111111(0dB)～000000(－31.5dB)のレベルで変化．
CH1-0　：チャネルを設定する．

ボリューム1

CH1	CH0	チャネル
0	0	ライン入力の左
0	1	ライン入力の右

ボリューム2

CH1	CH0	チャネル
0	0	CD音声の左出力
0	1	CD音声の右出力
1	0	マイク入力
1	1	モデム出力*

＊ はオプションのモデムカード使用時

▼表Ｉ-5-2　COMレジスタD5-0の値と減衰量の関係

D5-0 543210	減衰量 (dB)	D5-0 543210	減衰量 (dB)	D5-0 543210	減衰量 (dB)	D5-0 543210	減衰量 (dB)
000000	−31.5	000001	−31.0	000010	−30.5	000011	−30.0
000100	−29.5	000101	−29.0	000110	−28.5	000111	−28.0
001000	−27.5	001001	−27.0	001010	−26.5	001011	−26.0
001100	−25.5	001101	−25.0	001110	−24.5	001111	−24.0
010000	−23.5	010001	−23.0	010010	−22.5	010011	−22.0
010100	−21.5	010101	−21.0	010110	−20.5	010111	−20.0
011000	−19.5	011001	−19.0	011010	−18.5	011011	−18.0
011100	−17.5	011101	−17.0	011110	−16.5	011111	−16.0
100000	−15.5	100001	−15.0	100010	−14.5	100011	−14.0
100100	−13.5	100101	−13.0	100110	−12.5	100111	−12.0
101000	−11.5	101001	−11.0	101010	−10.5	101011	−10.0
101100	− 9.5	101101	− 9.0	101110	− 8.5	101111	− 8.0
110000	− 7.5	110001	− 7.0	110010	− 6.5	110011	− 6.0
110100	− 5.5	110101	− 5.0	110110	− 4.5	110111	− 4.0
111000	− 3.5	111001	− 3.0	111010	− 2.5	111011	− 2.0
111100	− 1.5	111101	− 1.0	111110	− 0.5	111111	0.0

▼表Ｉ-5-3　dB値と伝送比率の関係

dB	比率(倍)	dB	比率(倍)	dB	比率(倍)	dB	比率(倍)
0.0	1.000	− 0.5	0.944	− 1.0	0.891	− 1.5	0.841
− 2.0	0.794	− 2.5	0.750	− 3.0	0.708	− 3.5	0.668
− 4.0	0.631	− 4.5	0.596	− 5.0	0.562	− 5.5	0.531
− 6.0	0.501	− 6.5	0.473	− 7.0	0.447	− 7.5	0.422
− 8.0	0.398	− 8.5	0.376	− 9.0	0.355	− 9.5	0.335
−10.0	0.316	−10.5	0.299	−11.0	0.282	−11.5	0.266
−12.0	0.251	−12.5	0.237	−13.0	0.224	−13.5	0.211
−14.0	0.200	−14.5	0.188	−15.0	0.178	−15.5	0.168
−16.0	0.158	−16.5	0.150	−17.0	0.141	−17.5	0.133
−18.0	0.126	−18.5	0.119	−19.0	0.112	−19.5	0.106
−20.0	0.100	−20.5	0.094	−21.0	0.089	−21.5	0.084
−22.0	0.079	−22.5	0.075	−23.0	0.071	−23.5	0.067
−24.0	0.063	−24.5	0.060	−25.0	0.056	−25.5	0.053
−26.0	0.050	−26.5	0.047	−27.0	0.045	−27.5	0.042
−28.0	0.040	−28.5	0.038	−29.0	0.035	−29.5	0.033
−30.0	0.032	−30.5	0.030	−31.0	0.028	−31.5	0.027
−32.0	0.025						

5.3 PCM音源

FMTOWNSでは，音源として，FM音源の外にPCM音源を内蔵しています．そして，PCM音源LSIにはRF5C68を採用しています．

このPCM音源によって，サンプリングされた音声の再生が可能です．再生時の最大チャネル数は8個であり，同時に8音を発音することができます．

この節では，PCM音源のハードウェアと，サンプリングの仕組みおよび，再生の仕組みについて解説しています．

5.3.1 サンプリングの原理

PCM音源のPCMはPulse Code Modurationの略です．音声をデジタルデータに変換し，音源として使用します．音の標本を取るという意味から，サンプリング音源ともいいます．

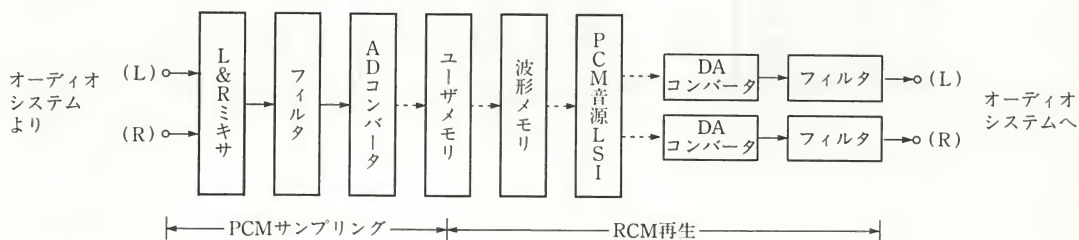
音のサンプリングとは，アナログの音声信号をデジタルデータに変換することを意味します．音の実体は振幅(空気の粗密)の時間的変化ですから，時間を細かい単位に分けその時々の振幅を2進値で記録すれば，音をデジタル化することができます．

サンプリングした音は，コンピュータのデータとして保存でき，また，データを時間的に操作することにより，再生時の周波数を変えることもできます．

5.3.2 PCM音源周辺のハードとその働き

PCM音源関係のブロックダイヤグラムを，図I-5-5に示します．

PCM音源に関係するハードウェアは，サンプリングを受け持つ部分とサンプリング音を再生する部分に分かれています．

▼図I-5-5 PCM音源ブロックダイヤグラム

各部のハードウェアについて説明します．

●L&R ミキサ

オーディオシステムから入力されたステレオ音声をモノラルに変えます．

●フィルタ

音声をデジタル化する際に不要な信号をカットします．フィルタは，サンプリングクロック（サンプリングレートに対応）と入力されたアナログ信号が干渉して不快な「うなり」音が発生するのを防止するとともに，サンプリングした音を汚くしてしまう原因となる高すぎる周波数成分を除去する働きを持っています．フィルタのカットオフ周波数は約 4KHz，−12dB/OCT の特性が得られるように設計されています．

●AD コンバータ

アナログ音声をデジタルデータに変換します．AD コンバータは１系統（モノラル）です．

●ユーザーメモリ

サンプリングされたデータを書き込みます．

●波形メモリ

再生するデータを格納しておくメモリです．再生時，波形メモリへの書き込みは CPU で行い，読み出しは PCM 音源 LSI が行いますが，CPU で読み出すことも可能です．

●PCM 音源 LSI

サンプリングデータを波形メモリから読み取って合成し，左右に振り分けて，DA コンバータに出力します．

●DA コンバータ

デジタルの波形データをアナログの音声データ（振幅）に変換します．

再生時には，PCM 音源 LSI は同時に８チャネル分の波形メモリの内容を読み出すことができます．この場合，各チャネルのデータは，DA コンバータへ出力される前に加算されます．

●フィルタ

PCM 音源出力に含まれている階段状の波形を滑らかにつなげるためのフィルタです．カットオフ周波数は，約 4KHz です．

5.3.3　サンプリングの仕組み

サンプリングは図 I-5-5の，L&R ミキサからユーザーメモリまでの間で行われます．

ここでは，サンプリングの中心となる AD コンバータによるアナログ音声のデジタル化とデータの記録方法について説明します．

●アナログ音声のデジタル化(波形データの作成)

アナログ音声のデジタル化はADコンバータで行います.

1秒間に行うサンプリングの回数をサンプリングレートといいます. サンプリングレートは最大19.2KHzで, 任意に回数を減らすことができます.

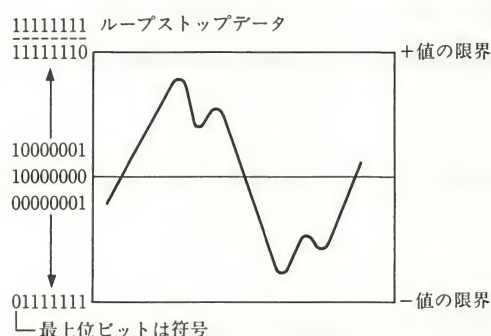
振幅の程度は254段階(−127〜+126)で読み取ります. 10000000(レベル0)を中心に, プラス方向に最大11111110(レベル+126), マイナス方向に最小01111111(レベル−127)までとなっています(図I-5-6).

サンプリングデータの最上位ビットは符号を表します. 最上位ビットが1ならば, レベルは正または0, 最上位ビットが0ならば, 負の値となります.

なお, データ中で次の2つは特別な意味があります.

11111111……………………ループストップデータ
00000000……………………ヌル(無効)

これらは, サンプリングデータとしては意味を持ちません.

▼図I-5-6 原信号とデジタル変換値の関係

● ADサンプリングデータレジスタへの波形の書き込み

ADコンバータの構造を概念的に示したのが, 図I-5-7です.

AD変換部で作成された8ビットのデジタルデータは, FIFO(ファーストインファーストアウト)を経由して, ADサンプリングデータレジスタ(表I-5-4)に書き込まれます.

FIFOは, AD変換部とADサンプリングデータレジスタの間のバッファの役目をしています.

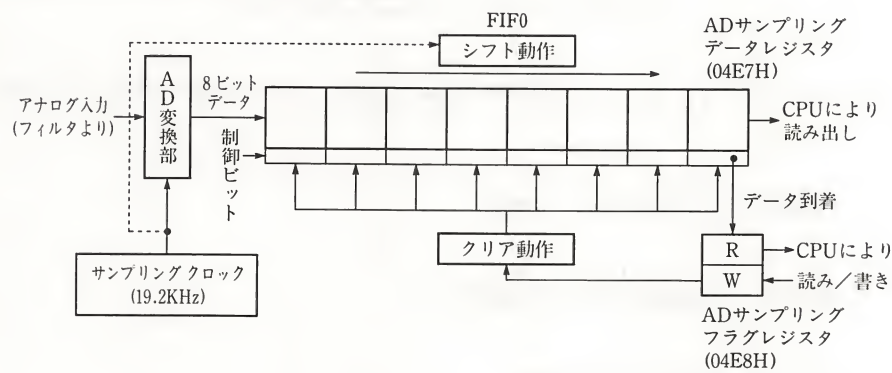
ADサンプリングデータレジスタの内容を読み取って, メモリに転送すれば, サンプリングデータがメモリに格納されることになります.

ADサンプリングデータレジスタの内容を読み出す際には, まず, FIFOの内容をクリアする

ために, AD サンプリングフラグレジスタ(表 I-5-5)に何かの値を書き込みます. これは, FIFO に波形データ以外のデータを残さないようにするためです. このとき書き込む値には特に意味はなく, そのデータも記憶されません. 00Hなどが適当です.

AD サンプリングデータレジスタの内容は, CPU によって読み出しの都度シフトされて更新されます. また, AD 変換部では, FIFO に空きがあれば書き込みを行い, シフト動作を行います.

AD サンプリングフラグレジスタの値は, CPU によってデータが読み出されると 0 になり, FIFO から値が書き込まれると 1 になります. したがって, AD サンプリングフラグレジスタの値が 1 になっていることを確認しながら, AD サンプリングデータレジスタを読み込めば, 正しく波形データを読み取ることができます.

▼図 I-5-7　AD コンバータの概念図

▼表 I-5-4　ADサンプリングデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04E7H	ADサンプリングデータレジスタ	R	SD7	SD6	SD5	SD4	SD3	SD2	SD1	SD0

▼表 I-5-5　ADサンプリングフラグレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04E8H	ADサンプリングフラグレジスタ	R	不定							サンプリングフラグ
		W	0	0	0	0	0	0	0	

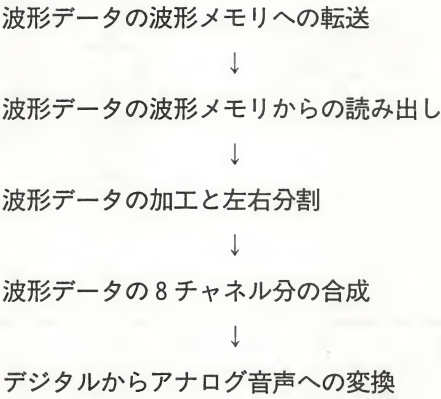
サンプリングフラグ (bit0)：ADSDにサンプリングされたデータが格納されていることを表す.
ADSDをリードすることにより, クリアされる.
ライトを行うとFIFOにあるすべてのサンプリングデータが失われる.

5.3.4 再生の仕組み

再生は図I-5-5の右側の部分，波形メモリ，PCM音源レジスタ，積分回路，フィルタの間で行われます．再生の過程では，PCM音源レジスタ（表I-5-6）を使用します．

▼表I-5-6　PCM音源レジスター覧

I/Oアドレス	名　称
04F0H	ENVデータレジスタ
04F1H	PANデータレジスタ
04F2H	FDLデータレジスタ
04F3H	FDHデータレジスタ
04F4H	LSLデータレジスタ
04F5H	LSHデータレジスタ
04F6H	STデータレジスタ
04F7H	コントロールレジスタ
04F8H	チャネルON/OFFレジスタ

ここでは，サンプリングした音を再生する仕組みを説明します．再生は次のような流れで行われます．

このうち，波形メモリへの転送はCPUの命令を使って行い，DA変換はDAコンバータが行いますが，その他は，PCM音源LSIが行います．

PCM音源LSIは，最大8チャネルの音を同時に再生することができます．複数のチャネルを使用する場合には，波形データの波形メモリへの転送，波形データの波形メモリからの読み出し，波形データの加工と左右分割などは，チャネルごとに行う必要があります．それぞれの設定をどのチャネルに対して行うかは，後述のコントロールレジスタ（表I-5-12）のCB2-0で指定します．

●波形データの波形メモリへの転送

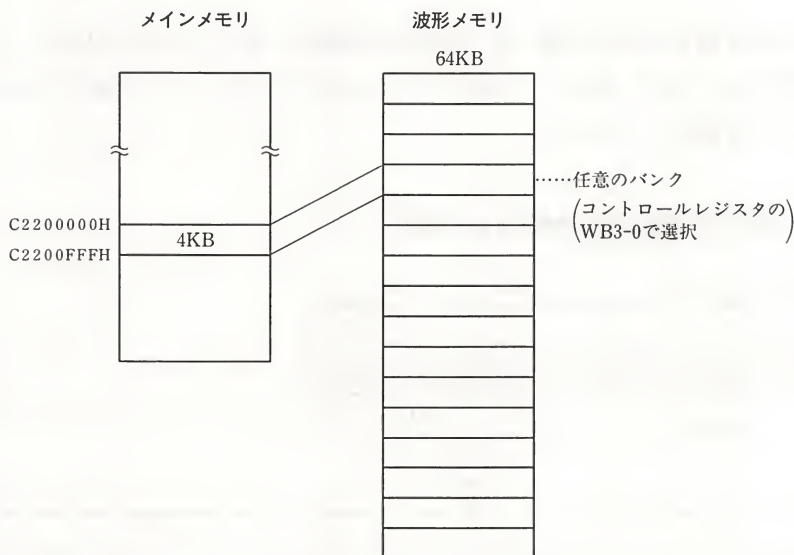
波形メモリは，再生する波形データを格納しておくメモリです．

PCM 音源 LSI は，波形メモリの内容を参照しながら，再生を行います．

したがって，再生を行うときには，あらかじめ，サンプリングによって作成した波形データを，波形メモリに転送しておく必要があります．

このメモリは，CPU から直接に読み書きすることはできず，メインメモリの C2200000H～C2200FFFH の範囲の 4KB を波形メモリのアクセスのための窓口として使用します(図Ⅰ-5-8)．すなわち，64KB の波形メモリを 4KB 単位に16のバンク(ブロック)に分け，バンクごとにメインメモリとの間で転送を行って，間接的に読み書きを行います．16のバンクのどれを読み書きするかは，後述のコントロールレジスタ(表Ⅰ-5-12)の WB3-0 によって決定されます．したがって，1つのサンプリングデータが 4KB を超える場合は，途中で WB3-0 の値を書き換える必要があります．

なお，複数のチャネルを使用する場合には，64KB の波形メモリを分割して使用しますので，それぞれに各チャネルで再生する波形を書き込んでおく必要があります．

▼図Ⅰ-5-8　波形メモリのアクセスの仕組み

●波形データの波形メモリからの読み出し

データを演奏する際の波形メモリの読み出しは，PCM 音源 LSI が行いますが，その前に ST データレジスタ(表Ⅰ-5-7)，FDL データレジスタ，FDH データレジスタ(表Ⅰ-5-8)などに，アドレスを指定する必要があります．

具体的には，読み出しの先頭アドレスをSTデータレジスタに，アドレスの更新値をFDLデータレジスタ，FDHデータレジスタに設定します．

PCM音源LSIは，この設定値に従って，アドレスの低い方から高い方へ向かってアドレスを更新しながら，波形メモリの内容を読み出していきます．このとき，STデータレジスタから与えられるのはアドレスの上位8ビットまでで，下位8ビットはすべて0となります．したがって，開始位置は256バイト単位に設定することになります．

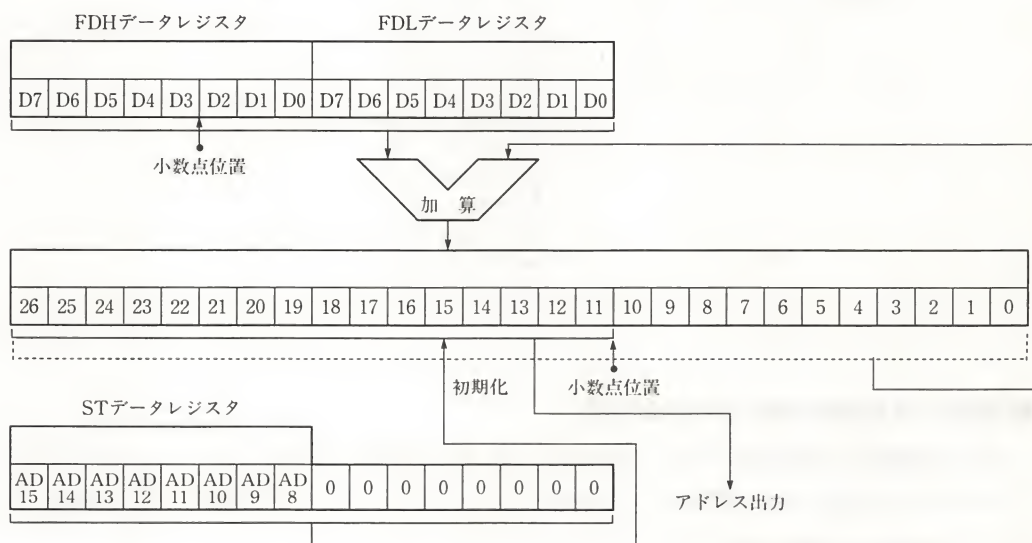
アドレスは27ビットで表されますが，アドレスの指定は26～11ビットまでの16ビット(整数部分)で行われ，10ビットから0ビットまでの小数点以下の部分は切り捨てられます．小数点以下の値を増加値として指定すると，増加値の加算が整数値になるまでは同じアドレスを読み出すことになります．

図Ⅰ-5-9は，波形メモリの読み出しの演算処理をまとめたものです．

アドレスの加算値をサンプリング時と同じにすると，サンプリングしたときと同じ波形を出力することができますが，加算値を変更すると再生速度，周波数を変えることができます．加算値を小さくすれば再生速度を遅く(周波数を低く)，大きくすれば再生速度を速く(周波数を高く)することができます．

この仕組みを利用して，１つの波形を元に，異なる周波数の音を再生させることが可能になります．

同じ波形を繰り返す場合には，繰り返しの最後の波形データとして11111111(ループストップデータ)を書き込んでおき，LSLデータレジスタ，LSHデータレジスタ(表Ⅰ-5-9)に繰り返しの先頭アドレスを設定しておきます．

▼図Ⅰ-5-9　波形メモリ読み出し時のアドレス演算

▼表Ⅰ-5-7　STデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F6H	STデータレジスタ	W	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8

波形メモリの読み出し時に，先頭アドレスを指定する．

▼表Ⅰ-5-8　FDL，FDHデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F2H	FDLデータレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0
04F3H	FDHデータレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0

波形メモリから読み出すアドレスの更新量を指定する．
各ビットのアドレス更新量の重みづけは次のようになる．

FDHデータレジスタ

ビット	重　み
D 7	2^{4}
D 6	2^{3}
D 5	2^{2}
D 4	2^{1}
D 3	2^{0}
D 2	2^{-1}
D 1	2^{-2}
D 0	2^{-3}

FDLデータレジスタ

ビット	重　み
D 7	2^{-4}
D 6	2^{-5}
D 5	2^{-6}
D 4	2^{-7}
D 3	2^{-8}
D 2	2^{-9}
D 1	1^{-10}
D 0	0^{-11}

▼表Ⅰ-5-9　LSL，LSHデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F4H	LSLデータレジスタ	W	AD7	AD6	AD5	AD4	AD3	AD2	AD1	AD0
04F5H	LSHデータレジスタ	W	AD15	AD14	AD13	AD12	AD11	AD10	AD9	AD8

波形メモリから読み出したデータがループストップデータ(11111111)のとき，LSLデータとLSHデータを波形メモリのアドレスにセットして再度読み出しを続ける．

●波形データの加工と左右分割

PCM音源LSIは，読み出された波形データに対して，振幅の拡大，縮小，左右への分割，上限値と下限値を超えるデータの抑制(リミッタ動作)などを行います．それには，ENVデータレジスタ(表I-5-10)，PANデータレジスタ(表I-5-11)を使用します．

波形データの加工は，図I-5-10のような演算処理によって行います．

▼表I-5-10　ENVデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F0H	ENVデータレジスタ	W	E N V							
			D7	D6	D5	D4	D3	D2	D1	D0

波形メモリから読み出したデータの振幅に強弱を与える．

▼表I-5-11　PANデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F1H	PANデータレジスタ	W	RIGHT				LEFT			
			PAN3	PAN2	PAN1	PAN0	PAN3	PAN2	PAN1	PAN0

発音中のチャネルから生成される出力を左右に分解する．

これを，個別に説明します．

振幅の拡大のためには，8ビットの波形データのうち符号部分を除く7ビットとENVデータレジスタの8ビットの値の掛け算をします．

次に，波形データの左右の定位を決めるため，PANデータレジスタの左右のそれぞれ4ビットの値と掛け算をすることにより，左右それぞれの波形の振幅を，19ビット値にします．PANデータレジスタの上位の4ビットは右チャネル，下位4ビットは左チャネルの音量配分を表します．したがって，左右どちらかのみを最大の音量にする場合は，次のようになります．

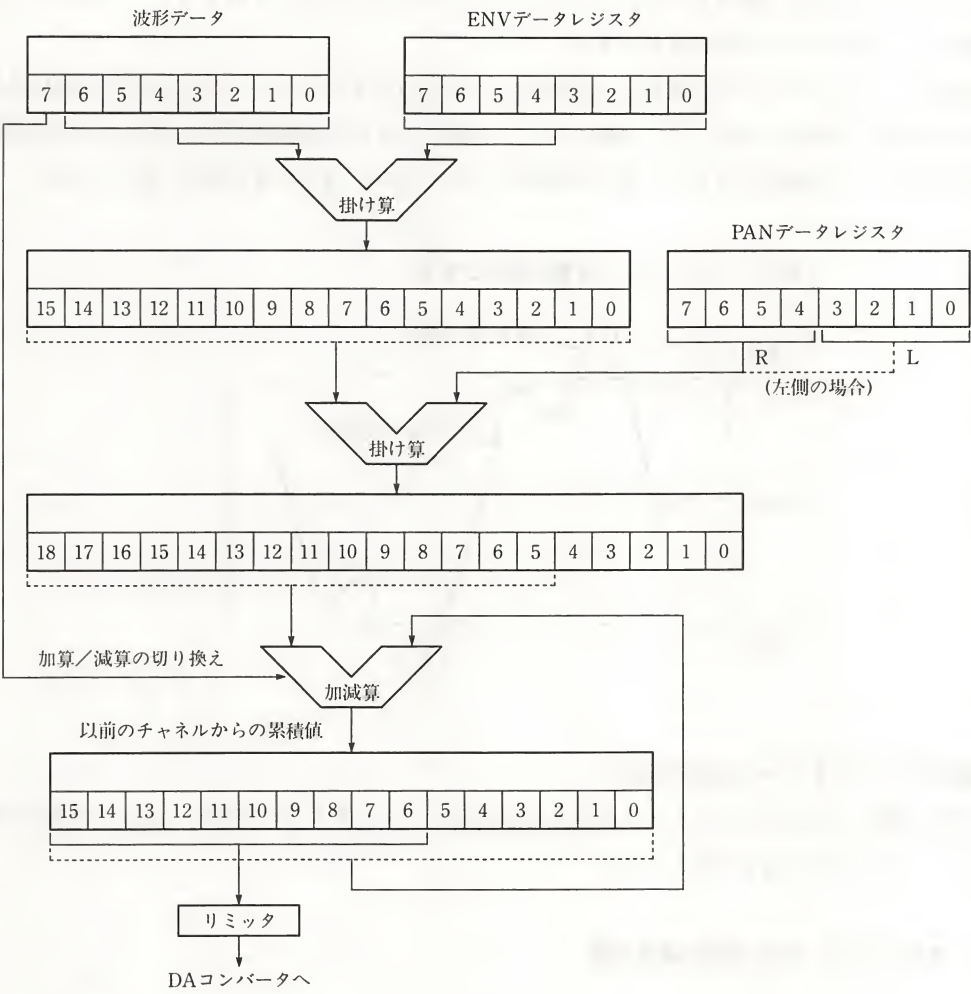
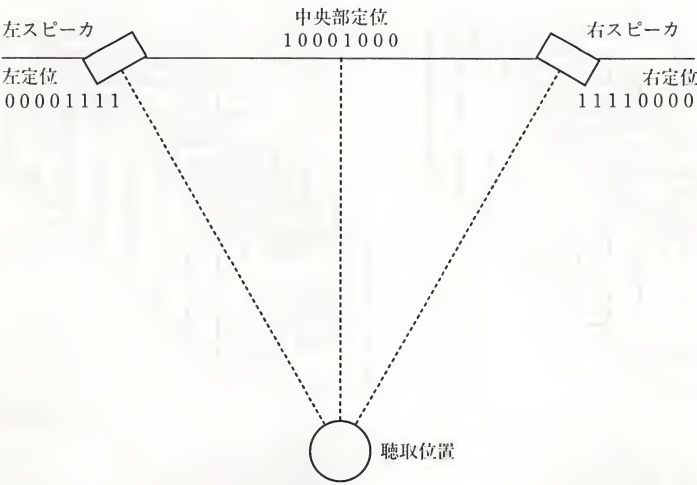
00001111……左側のみ
11110000……右側のみ

音を中央に定位させるためには左右の音量を同じにすればいいのですが，PANデータを11111111にすると，片側のときの2倍の音量になってしまいます．そこで，片方だけの音量と同程度の大きさにするには，10001000にします(図I-5-11)．

このように，左右の音の大きさを調整することにより，16方向の定位が得られます．また，再生中にPANデータを少しずつ変更すると，音が左右に移動するような効果を得ることができます

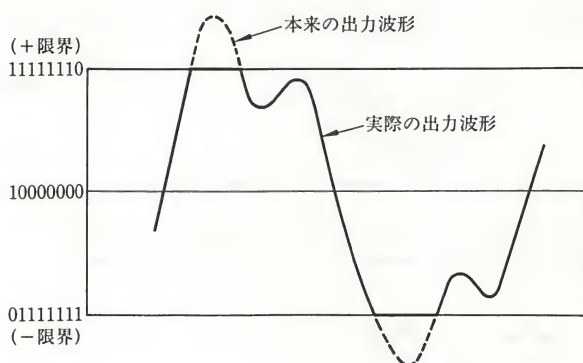
19ビット中の上位14ビットと以前のチャネルからの累積値を，加算または，減算します．加

▼図Ⅰ-5-10　波形データの演算処理

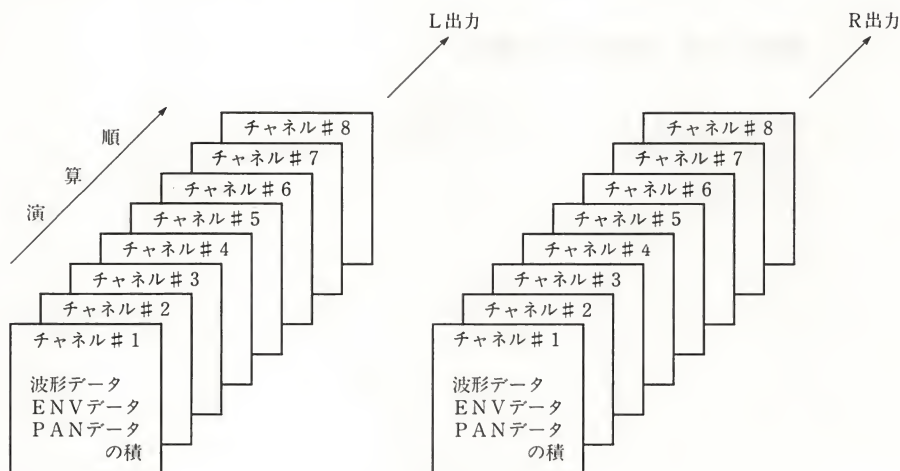
波形データ
7 6 5 4 3 2 1 0
ENVデータレジスタ
7 6 5 4 3 2 1 0
掛け算
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
PANデータレジスタ
7 6 5 4 3 2 1 0
R
L
(左側の場合)
掛け算
18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
加算／減算の切り換え
加減算
以前のチャネルからの累積値
15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
リミッタ
DAコンバータへ

▼図Ⅰ-5-11　PANデータと定位

算するか減算するかは，その波形データの符号によって決まります．

その後に，16ビット中の上位10ビットがリミッタに送られます．このように，下位ビットを省略すると波形データが縮小されます．

最後に，リミッタによる上限値と下限値のカットを行います．リミッタでは前の演算中に桁あふれが検出されたときは，プラス側，マイナス側ともそれぞれ限界値にします．この動作が行われると，その部分だけ正しく波形が再現できなくなり，音がひずみます(図I-5-12)．

▼図I-5-12　リミッタ動作時のひずみ

●波形データの8チャネル分の合成

PCM音源LSIは，左右の8チャネルずつの波形データをそれぞれ加算し，左右2系統のDAコンバータに出力します(図I-5-13)．

▼図I-5-13　PCM音源の演算処理

●デジタルデータからアナログ音声への変換

8チャネル分を合成した波形データを左右2系統のDAコンバータを使ってアナログ音声に変換します．

5.3.5 チャネルの選択とON/OFF

波形メモリの読み出しと波形データの加工を行う際に，それをどのチャネルに対して行うかの指定には，コントロールレジスタ（表I-5-12）を使います．またチャネルの使用の有無の設定には，チャネルON/OFFレジスタ（表I-5-13）を使います．

▼表I-5-12　コントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F7H	コントロールレジスタ	W	ON/OFF	MOD	0	0	WB3	WB2	WB1	WB0
							0	CB2	CB1	CB0

モードの設定や波形メモリのバンクアドレス，チャネルの設定を行う．

ON/OFF(bit7)　：PCM音楽の発音動作の制御をする．
0＝発音動作を開始しない
1＝発音動作を開始する（1のときCPUからのリードは不可能）

MOD(bit6)　：ビット3-0のモードを設定する．
0＝ビット3-0をWB3-0として使用する
1＝ビット2-0をCB2-0として使用する

WB3-0(bit3-0)　：波形メモリのバンクアドレスを指定する（1～16）．

CB2-0(bit2-0)　：チャネルアドレスを指定する（1～8）．

▼表I-5-13　チャネルON/OFFレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04F8H	チャネルON/OFFレジスタ	W	CH8	CH7	CH6	CH5	CH4	CH3	CH2	CH1

各チャネルの発音状態の制御を行う．各ビットが0のとき発音する．コントロールレジスタのビット7が1のときのみ有効．

●コントロールレジスタ

コントロールレジスタの最上位ビット（ON/OFFビット）は，PCM音源の再生動作を制御するもので，1を書き込むと再生が開始されます．再生中は，波形メモリはPCM音源が再生に使用するので，CPUからの読み出しが不可能になります．このレジスタの下位4ビットは，波形メモリのバンクの設定を行うビットとして使われます．

また，下位3ビットは，チャネル番号を設定するビットとしても使われます．

ここで設定されたチャネルは，ENVデータ，PANデータ，FDLデータ，FDHデータ，LSLデータ，LSHデータ，STデータの7つのデータレジスタのチャネルを指定するものです．このように，下位3ビットは2つの異なる意味がありますが，MODのビットを0にした場合は波形メモリのバンクの設定に，1にした場合はチャネル番号の設定に使われます．

再生時に個々のチャネルごとに，使用の有無を定義しているのが，チャネルON/OFFレジスタで，それぞれのビットが0のとき，そのチャネルが有効になります．この値が1になっていると，対応するチャネルの音を再生することはできません．

5.3.6 波形メモリの読み出し時の割り込み処理

PCM音源LSIの波形メモリの読み出しは，あらかじめ指定された先頭アドレスと増加値の設定に従って行われますが，PCM音源LSI自身には自動停止する手段は用意されていないので，放っておくと波形メモリの最後まで読んで，また始めから読み出すということを繰り返します．

これでは，波形メモリを連続して使用することができません．そこで，再生中に読み出した波形メモリのバイト数が4KBになったら割り込みを発生するようにしています．その結果，4KBの波形データを再生したら(読み込んだら)，CPUに制御を移し，次のバンクを読むか，または，PCM音源LSIの動作を止めるといったことができます．

波形メモリの64KBについては，先頭から，8KB単位に8個の区間に分け，それぞれの区間ごとに，割り込みを許可するかどうかを選択することができます．また，どの区間で割り込みが起こったかを知ることもできます．それぞれの区間を8つのチャネルの波形データの格納領域として使うとソフトウェアの負担が軽くなります．

● PCM割り込みマスクレジスタとPCM割り込みレジスタ

この割り込みの制御には，PCM割り込みマスクレジスタ(表I-5-14)とPCM割り込みレジスタ(表I-5-15)を使用します．

PCM割り込みマスクレジスタは，割り込みの許可／不許可を決めるもので，その区間に対応するビットを1に設定した場合には，割り込みを発生させるように，0の場合には割り込みを発生させないようになっています．

また，PCM割り込みレジスタは，どの区間で，割り込みが発生したかを参照するためのものです．割り込みが発生するとその区間に対応するビットが1になります．

なお，PCM割り込みレジスタは読み出した直後，全ビットが0にクリアされます．

PCM割り込みマスクレジスタ，PCM割り込みレジスタの各ビットと各区間のアドレスの範囲の対応関係は，表I-5-16のようになります．

割り込みは，他の原因によっても起こります．そのとき起こった割り込みが，PCM割り込みであるかどうかを調べるには，INT13割り込み要因レジスタ(表I-5-17)を参照します．PCM

ビットが1であれば，PCM割り込みであるということになります．したがって，先にINT13割り込みレジスタを参照して，PCM割り込みならばPCM割り込みレジスタを参照するという手順を踏みます．

▼表 I-5-14　PCM割り込みマスクレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04EAH	PCM割り込みマスクレジスタ	R/W	M7	M6	M5	M4	M3	M2	M1	M0

波形メモリのバンク単位に，割り込みのマスクをする．

▼表 I-5-15　PCM割り込みレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04EBH	PCM割り込みレジスタ	R	IF7	IF6	IF5	IF4	IF3	IF2	IF1	IF0

割り込みが発生している波形メモリのバンクを読み出す．

▼表 I-5-16　PCM割り込み関係のレジスタのビットと波形メモリのアドレスの関係

波形メモリのアドレス範囲	許可	割り込み
E000H ～ FFFFH	M7	1F7
C000H ～ DFFFH	M6	1F6
A000H ～ BFFFH	M5	1F5
8000H ～ 9FFFH	M4	1F4
6000H ～ 7FFFH	M3	1F3
4000H ～ 5FFFH	M2	1F2
2000H ～ 3FFFH	M1	1F1
0000H ～ 1FFFH	M0	1F0

▼表 I-5-17　INT 13割り込み要因レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04E9H	INT13割り込み要因レジスタ	R	不定				PCM	不定		FM

PCM (bit3)　：PCM割り込み有無．
0＝割り込みなし
1＝割り込みあり

FM (bit0)　：FM音源からの割り込み(タイマ等)有無．
0＝割り込みなし
1＝割り込みあり

5.4 FM音源

FMTOWNSでは，FM音源LSIにYM2612を採用しています．このFM音源LSIには6個のチャネルがあり，同時に6音を再生することができます．また，音を各チャネルごとに左右の両方，または左右に振り分けて出力することができます．

この節では，FM音源のハードウェアの構造および，音を発生する仕組みについて解説しています．

5.4.1 スロット

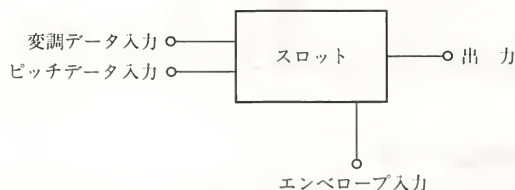
FM音源のFMはFrequency Moduration(周波数変調)の略で，時間軸成分の変調により発生する高調波を利用して新しい音を作ることをいいます．

音を合成するための最小の単位として，スロットがあります．

スロットは，波形合成(楽音を作る)の最小単位で，図I-5-14のように3つの入力端子と1つの出力端子があります．3つの入力端子から入力した波形信号を，電気的に加算，変換，および乗算して，出力端子に出力します．

スロットを決められたパターンで4つ接続したものが音源チャネルとなります．

なお，スロットに入力されたデータの加算，乗算などを行っている演算器をオペレータといいます．YM2612には4個のオペレータがあり，時分割で6チャネル分の演算を行っています．

▼図I-5-14 スロットの出力

●スロットの構造

スロットの論理的構造は図I-5-15のようになっています．

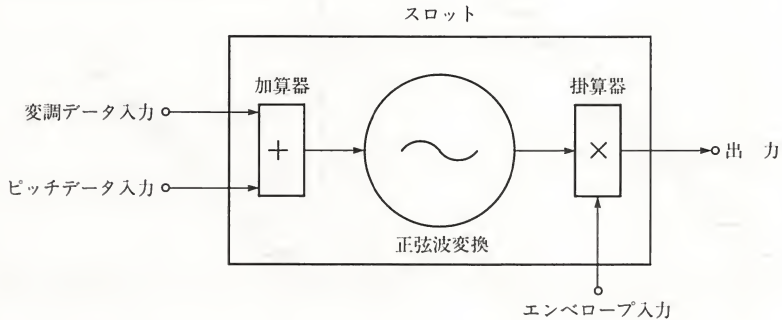
この加算器と掛算器は実際には，スロット内部にあるのではなくオペレータにあり，演算はそこで行われます．

ピッチデータ入力は，波形の周波数を決めるための信号で，音源IC内部で作られたノコギリ波が使われます．

変調データ入力は，波形の形状を変化させるための信号です．そのスロット自身の出力をフィードバックして使ったり，他のスロットの出力を使用します．

エンベロープ入力は，波形の振幅を時間的に変化させるための信号です．エンベロープジェネレータという回路で作られます．

ピッチデータ入力とデータ変調入力から入った2つの信号は，加算器によって合成されます．次に正弦波対応の変換がなされ，最後に，エンベロープ信号と乗算されて出力されます．

▼図 I-5-15　スロットの構造

5.4.2　スロットでの波形合成

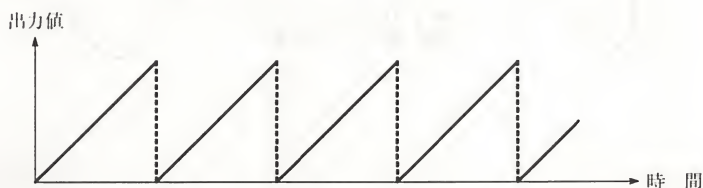
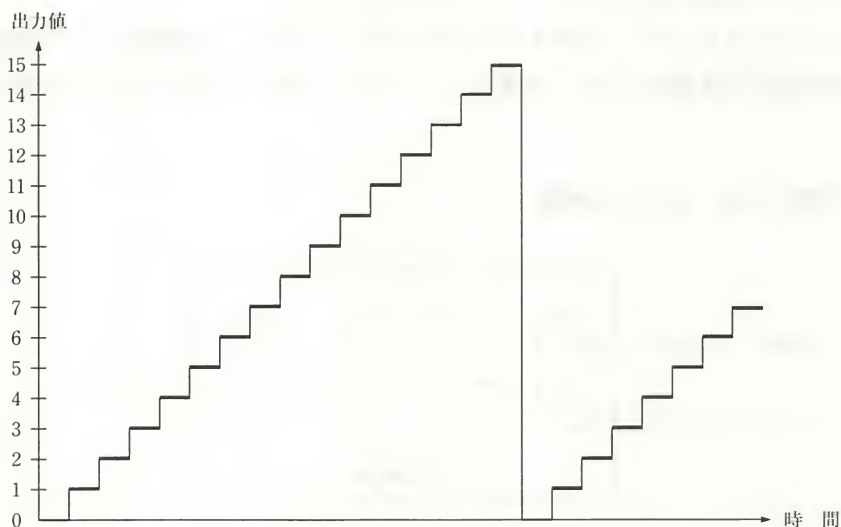
スロットの波形合成の仕組みについて解説します．

●ノコギリ波の生成と入力

ピッチデータ入力に使われるノコギリ波は，図 I-5-16のような形で，拡大すると図 I-5-17のようになります．

ノコギリ波は一定の時間間隔で出力されており，出力値はアナログ的にみると連続しておらず階段波とでもいうべき形になっています．

出力値の生成には，カウンタを使います．一定間隔のクロックパルスに合わせてカウントし，出力値を増加させることにより，ノコギリ波が生成します．フルカウントになったときに桁上がりを無視し，再び0からカウントアップを繰り返せば，同一周期のノコギリ波が生成できます．図 I-5-17はカウンタを4ビット(16段階)にした場合で，カウンタが15(1111)になると，次は5ビット目への桁上がり無視して0(0000)となります．

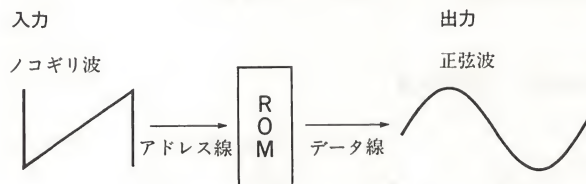
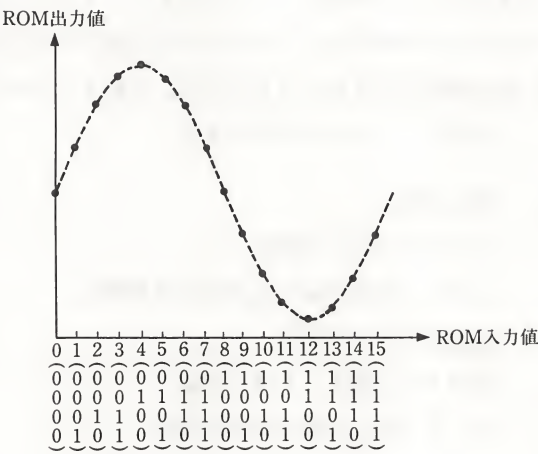
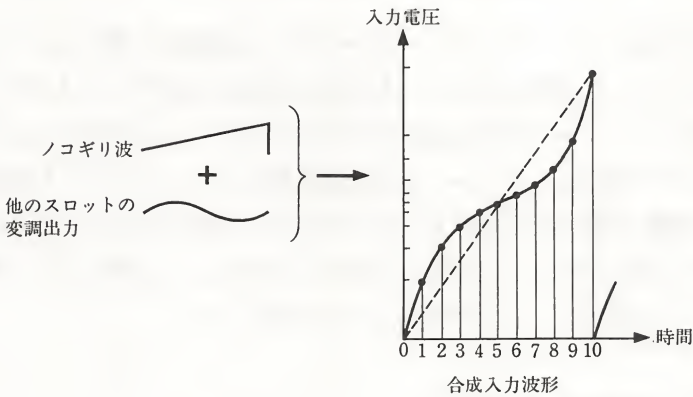
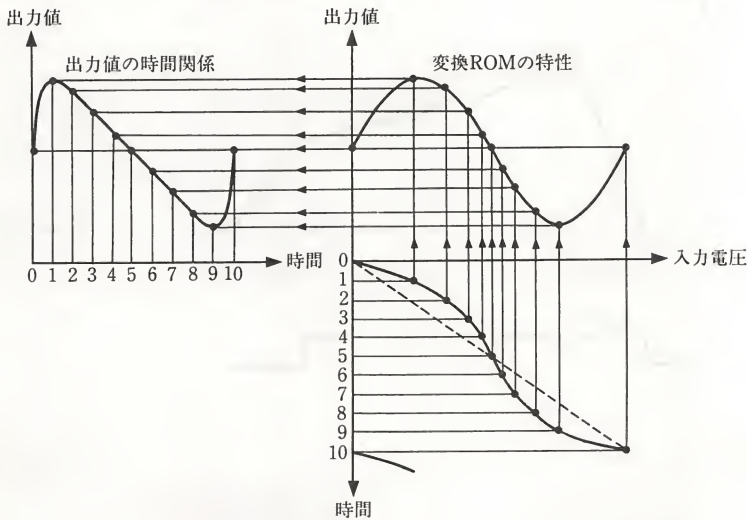
▼図 I-5-16　ノコギリ波

▼図I-5-17　ノコギリ波の拡大図

●波形の変換

変調データ入力とピッチデータ入力を合成した波形から，新しい波形への変換演算は，ROMをモデルにすると考えやすいので，以下，その前提で説明を進めます．

すなわち，ROMの各アドレスには変換後の正弦波の部分値が書き込んであるものとすれば，ROMに単純なノコギリ波を入力すると，きれいな正弦波が発生します(図I-5-18，図I-5-19)．

実際には，純粋なノコギリ波が入力されるとは限らず，一般に変調データ入力(他のスロットの変調出力)とピッチデータ入力(ノコギリ波)が合成され(図I-5-20)，合成された波形がROMのアドレス線に送られ，新しい波形を生成します(図I-5-21)．このような方法で生成される波形は，たくさんの高調波(倍音)を含み，自然の音に近いものとなります．

▼図I-5-18　ROMによるノコギリ波から正弦波への変換モデル

入力
ノコギリ波
アドレス線
ROM
データ線
出力
正弦波

▼図Ⅰ-5-19　ROMの入力値と出力値の関係の例

▼図Ⅰ-5-20　ノコギリ波と他のスロットの変調出力の合成

▼図Ⅰ-5-21　ROMによる波形の変換

●エンベロープ信号の入力

楽音には，ドラムのように立ち上りが鋭く，すぐに消えてしまうような音と，バイオリンを弓で引いたときのように立ち上りが穏やかで，なかなか消えないといった場合があります．このような楽音の違いは，音の振幅の変化によるものです．これをエンベロープといい，エンベロープを決める要素としては次のようなものがあります．

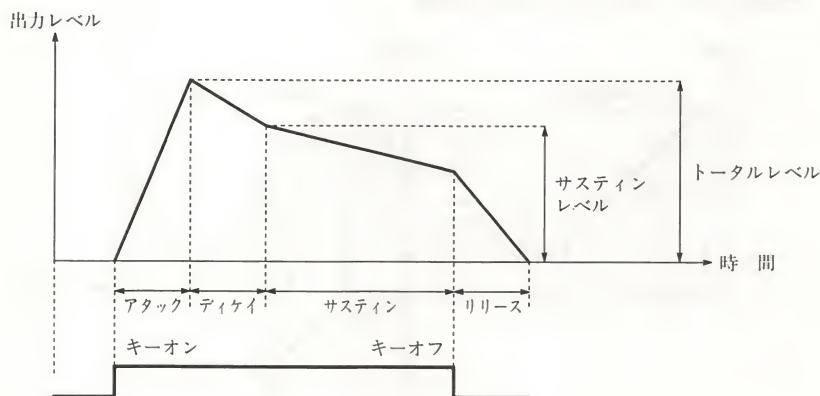
トータルレベル………最大振幅
アタック………………立ち上りの長さ(時間)
ディケイ………………立ち上り直後の落ち込みの長さ(時間)
サスティン……………後引きの長さ(時間)
サスティンレベル……後引きを開始するときの振幅
リリース………………キーオフ後の余韻の長さ(時間)

FM音源において，このようなエンベロープの効果をもたらすのが，エンベロープ入力信号です．

エンベロープ入力信号は，エンベロープジェネレータという回路から発生しますが，その際，図Ⅰ-5-22に示すエンベロープの要素，各レベル配分などを任意に設定することができます．

スロットは，個別にエンベロープ信号を接続するかどうかのスイッチを持っているので，スロットごとにエンベロープ変調(エンベロープ信号との乗算)をかけるかどうかを選択できます．

エンベロープは，振幅の時間的変化を与えるものですから，最後のスロットだけに与えるのが一般的ですが，それ以外のスロットで使用してもかまいません．その場合には，後続のスロットの入力信号のレベルが変化し，音色が時間とともに変化します．

▼図Ⅰ-5-22　FM音源のエンベロープ

5.4.3 スロットの接続

前述のように，１つのチャネルは４つのスロットによって構成されており，接続の仕方によって音色が変わります．ここでは，接続方法について説明します．

●スロット接続のアルゴリズム

スロットの接続方法は，表Ｉ-5-18に示す８とおりの組み合わせの中から選択することができます．これをアルゴリズムといいます．

スロットが直列に接続されている度合が高いものほど複雑な波形が生成され高調波を多く含んでいるため明るい感じの音になります．

また，スロットが並列に接続されている度合が高いものは，三重，四重に音が並列に合成されるので，重厚な味わいのある音となります．例えば，オルガンなどの感じを出す場合とか，複数の楽器を演奏しているような感じを出す場合に有効です．

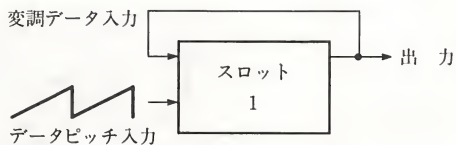
パターン４は，中間的なもので，２系統の単純な合成で結果が予測しやすいため，音作りが比較的容易に行えます．

●セルフフィードバック

各チャネルのスロット１については，自己の出力が変調データ入力に接続されています(図Ｉ-5-23)．これが，セルフフィードバックです．

フィードバックの程度は任意に設定できます．

フィードバックを行うと，正弦波とノコギリ波が合成されたときのようになります．高調波をたくさん含むので，刺激的な感じの強い音になります．

▼図Ｉ-5-23　セルフフィードバック

▼表I-5-18　スロットのアルゴリズム

Pはピッチデータ入力，Sはスロットを表す．

5.4.4　音のゆらぎについて

音のゆらぎは，音程，音の強弱などを故意にずらすことによって生まれます．FMTOWNSのFM音源では，音程のゆらぎは，LFO(超低周波数発振器)を使って，ハードウェアレベルで，作り出すことができます．

音程のゆらぎを作るには，ディチューンとマルチプルによって周波数を変化させる方法がありますが，FMTOWNSでは，ディチューンとマルチプルを使う必要はなく，これらを使うとすれば，固定ずれ(故意に調子をはずさせる)を発生させるときなどが考えられます．

なお，従来のFMシリーズの８ビット機に使われてきたFM音源(YM2203)では，LFOがないため，ディチューンとマルチプルをソフトウェアレベルでコントロールしていました．

5.4.5　チャネル３の特別な設定

チャネル３は，各スロットごとに周波数(ピッチデータ入力の周波数)を設定できます．詳しくは，FM音源全体の制御に関わる内部レジスタの項を参照してください．

5.4.6　FM音源の内部レジスタ

FMTOWNSが採用しているFM音源(YM2612)には２組の内部レジスタがあります．

１組をチャネル１～３，もう１組をチャネル４～６の制御に使用します．

チャネル１～３の制御用のレジスタのビット構成を表Ｉ-5-19に示します．チャネル４～６までの制御を行うレジスタのビット構成も，内部アドレスの30H～B6Hまでは同じです．

21H～2CHは，FM音源全体に関わる制御を，30Hから9EHは，スロット単位の制御を行います．

30H～9EHは，複数あるチャネルとスロットのそれぞれに対する制御を行いますが，内部アドレスとチャネル，スロットの対応関係は表Ｉ-5-20のようになっています．途中に欠番があり，スロット番号の順が１，３，２，４となっていることに注意してください．

▼表Ｉ-5-19　FM音源の内部レジスタ

内部アドレス	ビット構成								説明	備考
	b7	b6	b5	b4	b3	b2	b1	b0		
21H	TEST								LSIのTEST DATA	音源全体
22H					LFO				LFOのFREQ CONTROL	
24H	TIMER-A								TIMER-Aの上位８ビット	
25H							TIMER-A		TIMER-Aの下位２ビット	
26H	TIMER-B								TIMER-Bのデータ	
27H	MODE		RESET B A		ENABL B A		LOAD B A		TIMER-A/BのControlおよび3CHのMODE	
28H	SLOT					CH			KEY ON/OFF	

内部アドレス	ビット構成								説明	備考
	b7	b6	b5	b4	b3	b2	b1	b0		
2AH									予約済	音源全体
2BH										
2CH	TEST								LSIのTEST DATA	
30H〜3EH		DT			MULTI				Detune/Multiple (33, 37, 3Bのアドレスは欠番)	スロット単位
40H〜4EH		TL							Total Level (43, 47, 4Bのアドレスは欠番)	
50H〜5EH	KS			AR					Key Scale/Attack Rate (53, 57, 5Bのアドレスは欠番)	
60H〜6EH	AM			DR					AMON/Decay Rate (63, 67, 6Bのアドレスは欠番)	
70H〜7EH				SR					Sustain Rate (73, 77, 7Bのアドレスは欠番)	
80H〜8EH	SL				RR				Sustain Level/Release Rate(83, 87, 8Bのアドレスは欠番)	
90H〜9EH					SSG-EG				SSG-Type Envelope Control(93, 97, 9Bのアドレスは欠番)	
A0H A1H A2H	F-Num. 1								F-Number/BLOCK	チャネル単位
A4H A5H A6H			BLOCK			F-Num. 2				
A8H A9H AAH	3CH * F-Num. 1								3CH-3Slot F-Number/BLOCK	チャネル3の特別仕様
ACH ADH AEH			3CH * BLOCK			3CH * F-Num. 2				
B0H B1H B2H			FB			CONNECT			Self-Feedback/Connection	チャネル単位
B4H B5H B6H	L	R	AMS			PMS			出力/ゆらぎ設定	

▼表 I-5-20　内部アドレス30H～9EHのチャネル，スロットの対応関係

内部アドレス	チャネル	スロット
×0H ×1H ×2H	1 (4) 2 (5) 3 (6)	1
×4H ×5H ×6H	1 (4) 2 (5) 3 (6)	3
×8H ×9H ×AH	1 (4) 2 (5) 3 (6)	2
×CH ×DH ×EH	1 (4) 2 (5) 3 (6)	4

×3H, ×7H, ×BHは欠番(×は3～9)の任意の数.

● FM音源レジスタのアクセスのためのI/O

内部レジスタへの書き込みには，表I-5-21に示す4つのレジスタを使用します．

FM音源レジスタへの読み書きは，FM音源アドレスレジスタに内部アドレスの番号を指定し，続けて，FM音源データレジスタにデータを書き込む(あるいは読み出す)ようにします．

なお，FM音源の内部レジスタには，チャネル1～3までとチャネル4～6までの2系統があるので，FM音源アドレスレジスタとFM音源データレジスタも2系統用意されています．

FM音源ステータスレジスタは，レジスタに書き込みできない状態を示すBUSY(1のとき禁止)とタイマのフラグから成っています．タイマA，Bのフラグはそれぞれカウントオーバーを参照するためのものです．

▼表 I-5-21　FM音源制御のためのレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04D8H	FM音源ステータスレジスタ	R	BUSY	不　定					FLAG B	FLAG A
	FM音源アドレスレジスタ0	W	タイマ，チャネル1～3　アドレス							
			A7	A6	A5	A4	A3	A2	A1	A0
04DAH	FM音源データレジスタ0	W	タイマ，チャネル1～3　ライトデータ							
			D7	D6	D5	D4	D3	D2	D1	D0
04DCH	FM音源アドレスレジスタ1	W	チャネル4～6　アドレス							
			A7	A6	A5	A4	A3	A2	A1	A0
04DEH	FM音源データレジスタ1	W	チャネル4～6　ライトデータ							
			D7	D6	D5	D4	D3	D2	D1	D0

5.4.7 FM音源全体の制御にかかわる内部レジスタ

ここでは，内部レジスタのうち，番号(アドレス)21H～2CHのものについて説明します．

21H，2CHはメーカーがテストするためのもので，ユーザーがプログラムを作成する場合には，全ビット0にしておきます．

●LFOの設定

LFOは音程(または強弱)のゆらぎを作ります．ビット構成を表I-5-22に示します．LFOを使用する場合には，LFOビットを0にし，周波数をFREQ-CTRLに設定します．

指定した周波数の値が大きいほど，単位時間当たりのゆらぎの回数が多くなります．

▼表I-5-22 LFO設定のレジスタ

内部アドレス	7	6	5	4	3	2	1	0
22H					LFO	FREQ-CTRL		

LFO(bit3) ： 1のとき，LFO ON.
FREQ-CTRL (bit2-0) ： 周波数を下記のように設定する．

FREQ-CTRL	0	1	2	3	4	5	6	7
freq(Hz)	3.98	5.56	6.02	6.37	6.88	9.63	48.1	72.2

●タイマの設定

タイマ(Timer)は，キーオンからキーオフまで時間をカウントする目的で使われます．これにより，フルカウントでタイマは割り込みを起動するので，CPUに音の長さをカウントさせる必要がなくなり，他の仕事ができるようになります．

すなわち，キーオンのときタイマをセットし，キーオフの時間で割り込みがかかるようにします．そしてCPUからキーオフを行うことで音長が制御できます．

複数の楽器の音を同時に発音するとき，音長が異なると1個のタイマでは制御が面倒ですが，FM音源ではタイマが2個あるので，この問題が解決されています．タイマAは短時間用，Bが長時間用になっているので，タイマBが空いているとき，マウスの読み取り用に使われることもあります．

タイマの設定には，レジスタ番号24H～26Hのレジスタ(表I-5-23)を使用します．

タイマはAが10ビット，Bが8ビットのプリセッタブル型(時間を事前にセットするタイプ)です．Aの10ビットは24H(上位8ビット)，25H(下位2ビット)のレジスタを連結して初期値(NA)を構成します．Bについては26Hのレジスタ値が直接初期値(NB)となります．

タイマは初期値に一定の間隔で1を加算し，最上位から桁上がりを発生した段階で指定があ

れば，FM 音源ステータスレジスタにフラグを立て（１にする）ます．このとき割り込みが発生します．

時間はそれぞれ，

$$TA = \frac{12 \times (1024 - NA)}{\text{内部クロック周波数 [KHz]}} \quad \text{[ms]}$$

$$TB = \frac{192 \times (256 - NB)}{\text{内部クロック周波数 [KHz]}} \quad \text{[ms]}$$

で計算できます．内部クロック周波数は600KHz です．

タイマの制御は，レジスタ番号27Hのレジスタ（表 I-5-24）で行います．

▼表 I-5-23　タイマの設定を行うレジスタ

内部アドレス	7	6	5	4	3	2	1	0
24H	TIMER-A							
25H						TIMER-A		
26H	TIMER-B							

▼表 I-5-24　タイマの動作設定などのレジスタ

内部アドレス	7	6	5	4	3	2	1	0
27H	MODE		リセット		フラグ許可		ロード	
			B	A	B	A	B	A

MODE（bit7-6）　：チャネル３のモードを設定する．

MODE	モード	意　味
0　0	通　常	チャネル３は通常の発音モードになる．発生周波数はA2H，A6Hで設定する．
1　0	効果音	チャネル３は効果音モードになる．発生周波数は各スロットごとに設定できる．１スロットはA9H，ADH，２スロットはAAH，AEH，３スロットはA8H，ACH，４スロットはA2H，A6Hで設定する．
0　1	音声合成	発生周波数の設定は効果音のモードの場合と同様だが，チャネル３のキーのオンオフはタイマAでコントロールされ，タイマAのロードでキーオンとなり，カウント終了で，キーオフとなる．

リセット（bit5-4）　：１＝フラグをリセットする
フラグ許可（bit3-2）：１＝カウントオーバーのときフラグを１にする
ロード（bit1-0）　：１＝タイマに設定値をロードしてカウントを始める

NA，NBの初期値は，ロードビットの指定を1にすることによって，それぞれカウンタA，カウンタBに与えられ，カウントが開始されます．

フラグ許可ビットが1になっていると，カウンタ最上位の桁上がり発生時に，FM音源ステータスレジスタの該当側(AまたはB)に1がセットされます．フラグをクリアするには，リセットビットを1にすればよいのですが，この動作はフラグに対する応答の意味があるので，直後にリセットビットそのものはただちに消されます．

●モードの指定

モード(MODE)はチャネル3を特別な状態で使用するときに指定します．モードの指定にはレジスタ番号27Hのレジスタのビット7，6を使用します．

●キーのオンオフ制御

キーのオンオフ制御は各チャネルごとに設定することができます．内部レジスタ28H(表I-5-25)を使用します．キーのオンオフとは，楽音のオンオフを意味します．キーがオフになっても，リリース期間中は音が残ることに注意してください．

下位3ビットで，1～6のチャネルを，上位3ビットで1～4のスロットを選択します．例えば，4チャネルの1，3スロットをキーオンにしたい場合には，0101×100に設定します．ビット3は0でも1でもかまいません．

なお，2AH，2BHのレジスタは，システムで予約済になっており，ユーザーは使用することはできません．

▼表I-5-25 キーのオンオフ制御を行うレジスタ

内部アドレス	7	6	5	4	3	2	1	0
28H	SLOT					CH		
	4	3	2	1				

キーのオンオフに関して，スロットとチャネルの選択を行う．

SLOT(bit7-4)　：該当のビットを1にするとそのスロットがONになる．
CH(bit2-0)　：チャネルを選択する．

CH	チャネル番号
0 0 0	1
0 0 1	2
0 1 0	3
1 0 0	4
1 0 1	5
1 1 0	6

5.4.8 スロット単位に設定する内部レジスタ

ここでは，内部レジスタのうち，スロット単位に設定を行う，レジスタ番号30H～9EH のものについて説明します．

●ディチューン，マルチプルの設定

スロットに入力されるピッチデータ入力(ノコギリ波)の周波数を変化させ，故意に音程のずれを生じさせることができます．

ディチューン(DETUNE)は，各スロットの周波数をわずかにずらします．マルチプル(MULTIPLE)は，ディチューンの値を整数比倍して音をずらします．

この設定は，内部レジスタ30Hから3EH(33H，37H，3BH は除く)で行います．

ディチューンとマルチプルのビット構成は表 I-5-26のようになっています．

ディチューンの最上位ビット(b6)は符号を表し，0ならば正(加算)，1ならば負(減算)として扱われます．ディチューンの値と音階のずれとの対応は，表 I-5-27のとおりで，音階ブロックや音階により変化します．

▼表 I-5-26　ディチューンとマルチプルの設定を行うレジスタ

内部アドレス	7	6	5	4	3	2	1	0
30H～3EH	／	DETUNE			MULTIPLE			

33H，37H，3BHは除く．DETUNEはビット6が符号，ビット5，4が絶対値(0～3)を表す．

DETUNE (bit6-4)　：音ずれを設定する．設定値と音ずれの関係は表 I-5-25を参照．

MULTIPLE (bit3-0)　：音ずれの倍率を設定する．

MULTIPLE	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
倍　率	½	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

▼表Ｉ-5-27 DETUNE値と音階のずれの対応

BLOCK	NOTE	セント				周波数			
		DETUNE(下位2ビット)				DETUNE(下位2ビット)			
		0	1	2	3	0	1	2	3
0	0	0.000	0.000	4.491	8.970	0.000	0.000	0.048	0.095
0	1	0.000	0.000	3.784	7.580	0.000	0.000	0.048	0.095
0	2	0.000	0.000	3.479	6.353	0.000	0.000	0.048	0.095
0	3	0.000	0.000	2.674	5.343	0.000	0.000	0.048	0.095
1	0	0.000	2.247	4.491	4.491	0.000	0.048	0.095	0.095
1	1	0.000	1.891	3.780	5.667	0.000	0.048	0.095	0.143
1	2	0.000	1.590	3.179	4.767	0.000	0.048	0.095	0.143
1	3	0.000	1.337	2.874	4.009	0.000	0.048	0.095	0.143
2	0	0.000	1.124	2.247	4.491	0.000	0.048	0.095	0.191
2	1	0.000	0.346	2.836	3.780	0.000	0.048	0.143	0.191
2	2	0.000	0.795	2.395	3.179	0.000	0.048	0.143	0.191
2	3	0.000	0.663	2.006	3.342	0.000	0.048	0.143	0.238
3	0	0.000	1.124	2.247	2.808	0.000	0.095	0.191	0.238
3	1	0.000	0.346	1.891	2.836	0.000	0.095	0.191	0.286
3	2	0.000	0.795	1.590	2.385	0.000	0.095	0.191	0.288
3	3	0.000	0.668	1.672	2.340	0.000	0.095	0.238	0.334
4	0	0.000	0.562	1.405	2.247	0.000	0.085	0.238	0.381
4	1	0.000	0.709	1.418	1.891	0.000	0.143	0.286	0.381
4	2	0.000	0.597	1.193	1.769	0.000	0.143	0.298	0.429
4	3	0.000	0.502	1.170	1.672	0.000	0.143	0.334	0.477
5	0	0.000	0.562	1.124	1.545	0.000	0.191	0.381	0.525
5	1	0.000	0.473	0.946	1.418	0.000	0.191	0.381	0.572
5	2	0.000	0.398	0.895	1.232	0.000	0.191	0.429	0.620
5	3	0.000	0.418	0.836	1.170	0.000	0.238	0.477	0.668
6	0	0.000	0.351	0.773	1.124	0.000	0.238	0.525	0.763
6	1	0.000	0.555	0.709	1.005	0.000	0.286	0.672	0.811
6	2	0.000	0.298	0.646	0.945	0.000	0.286	0.520	0.906
6	3	0.000	0.283	0.585	0.835	0.000	0.334	0.668	0.354
7	0	0.000	0.281	0.562	0.773	0.000	0.391	0.763	1.049
7	1	0.000	0.236	0.473	0.650	0.000	0.391	0.763	1.049
7	2	0.000	0.199	0.399	0.547	0.000	0.391	0.763	1.049
7	3	0.000	0.167	0.334	0.480	0.000	0.381	0.763	1.049

NOTE＝N4×2＋N3
ここで，
　N4＝F11
　N3＝F11・(F10＋F9＋F8)＋$\overline{\text{F11}}$・F10・F9・F8
F8～F11はF-numberのビット番号を示す．

●トータルレベルの設定（スロットの音量レベルの設定）

トータルレベル（Total Level）は，音が立ち上がってピークに達したときのレベルです．表Ｉ-5-28はトータルレベルのビット構成です．各ビットがそれぞれ減衰量を表します．

例えば，ビット３と４が１のときは，

12＋６＝18［dB］

となります．この値が大きいほど音量が小さくなります．

トータルレベルの変更は，通常は楽音がキーオフになり，音が消えてから行う必要があります．そうしないと図Ｉ-５-24のように不自然なエンベロープ変化となります．ただし，音声合成モードの場合は次のキーオンから有効になるので，このような心配はありません．

▼表Ｉ-5-28　トータルレベルの設定を行うレジスタ

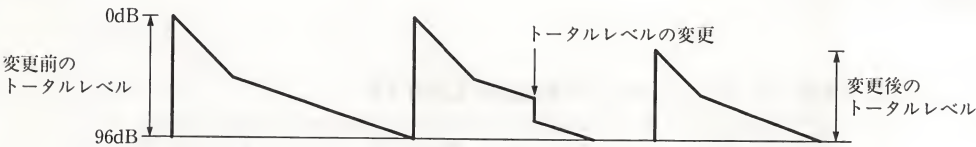
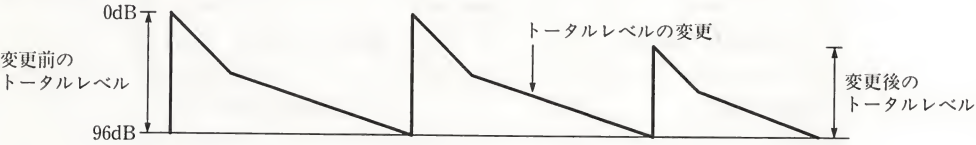
レジスタ番号	7	6	5	4	3	2	1	0
40H～4EH	／	TOTAL LEVEL						

43H，47H，4BHは除く．

TOTAL LEVEL（bit6-0）　：減衰量を指定する．

ビット	6	5	4	3	2	1	0
減衰量（dB）	48	24	12	6	3	1.5	0.75

▼図Ｉ-5-24　トータルレベル変更のタイミング

通常のエンベロープ

音声合成モードのエンベロープ

●エンベロープの設定（スロットの音量変化の設定）

エンベロープを設定するパラメータのビット構成は，表Ⅰ-5-29のようになります．

キースケール(KS)は音程によって，エンベロープの変化に差を付けるためのもので，エンベロープの各要素のレートを，音程によって変化させます．一般に高い音ほど変化が速いため，このような操作を行います．変換後のレートと原レートの関係は次のとおりです．

変換後レート＝原レート×２＋表Ⅰ-５-30の参照値

表Ⅰ-5-30のキーコードは，表Ⅰ-5-27のNOTEの値にBLOCK値を4倍した値を加えたもので，音程を反映した値です．なお，例外的に原レートが0のときは変換後レートも0になります．

アタックレート(Attack Rate)は立ち上り，ディケイレート(Decay Rate)は立ち上り直後の落ち込み，サスティンレート(Sustain Rate)は後引き，リリースレート(Release Rate)は余韻の時間を規定します．

これらの設定値と時間の関係は表Ⅰ-5-31のとおりです．

この表中，0db－96dbと10％－90％というのは，いずれも変化に要する時間とレート値の関係を表したものです．

サスティンレベル(Sustain Level)はディケイによって最終的に落ち込むレベルをトータルレベルとの相対値で表します．各ビットがそれぞれ表Ⅰ-5-32の減衰量を表します．表Ⅰ-5-31を参考に設定値を決めます．

リリースレート(Release Rate)は4ビットの指定値を2倍して，さらに1を加えた値をレート値として表Ⅰ-5-31を参照して設定値を決めます．

レジスタ番号6BH～6EHの7ビット目は，振幅変調をかけるかどうかの設定で，スロット単位にON/OFFするものです．1でON，0でOFFとなります．周波数変調，振幅変調の設定を参照してください．

▼表Ⅰ-5-29　エンベロープを設定するレジスタ

レジスタ番号	7	6	5	4	3	2	1	0
50H～5EH	KS			ATTACK RATE				
60H～6EH	AMON			DECAY RATE				
70H～7EH				SUSTAIN RATE				
80H～8EH	SUSTAIN LEVEL				RELEASE RATE			

×3H，×7H，×BHは除く．

▼表Ⅰ-5-30　KSの値とレート変換時の加算値

KS値	キーコード															
	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
1	0	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3
2	0	0	1	1	2	2	3	3	4	4	5	5	6	6	7	7
3	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

KS値	キーコード															
	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
0	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3
1	4	4	4	4	5	5	5	5	6	6	6	6	7	7	7	7
2	8	8	9	9	10	10	11	11	12	12	13	13	14	14	15	15
3	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31

キーコード＝BLOCK× 4 ＋NOTE

▼表Ⅰ-5-31　各レート値と時間の関係

アタックタイム			サスティン/ディケイ/リリースタイム			アタックタイム			サスティン/ディケイ/リリースタイム		
レート		mSEC (0dB-96dB)	レート		mSEC (0dB-96dB)	レート		mSEC (10%-90%)	レート		mSEC (10%-90%)
15	3	0.00	15	3	7.57	15	3	0.00	15	3	1.52
15	2	0.00	15	2	7.57	15	2	0.00	15	2	1.52
15	1	0.60	15	1	7.57	15	1	0.26	15	1	1.52
15	0	0.60	15	0	7.57	15	0	0.26	15	0	1.52
14	3	0.72	14	3	8.65	14	3	0.32	14	3	1.73
14	2	0.84	14	2	10.09	14	2	0.37	14	2	2.03
14	1	1.01	14	1	12.11	14	1	0.46	14	1	2.43
14	0	1.26	14	0	15.14	14	0	0.61	14	0	3.05
13	3	1.37	13	3	17.30	13	3	0.70	13	3	3.50
13	2	1.60	13	2	20.18	13	2	0.92	13	2	4.07
13	1	1.92	13	1	24.22	13	1	0.99	13	1	4.80
13	0	2.40	13	0	30.27	13	0	1.25	13	0	5.09
12	3	2.50	12	3	34.59	12	3	1.40	12	3	6.96
12	2	2.32	12	2	40.56	12	2	1.65	12	2	8.10
12	1	3.50	12	1	48.43	12	1	1.94	12	1	9.75
12	0	4.38	12	0	60.54	12	0	2.41	12	0	12.18
11	3	5.01	11	3	69.19	11	3	2.77	11	3	13.92
11	2	5.84	11	2	80.72	11	2	3.25	11	2	16.20
11	1	7.01	11	1	96.86	11	1	3.91	11	1	19.50
11	0	8.76	11	0	121.08	11	0	4.97	11	0	24.36
10	3	10.01	10	3	139.38	10	3	5.54	10	3	27.84
10	2	11.68	10	2	161.44	10	2	6.50	10	2	32.40
10	1	14.02	10	1	193.73	10	1	7.82	10	1	39.00
10	0	17.52	10	0	242.16	10	0	9.74	10	0	48.72
9	3	20.02	9	3	276.75	9	3	11.09	9	3	55.68
9	2	23.35	9	2	322.98	9	2	13.01	9	2	64.80
9	1	29.03	9	1	387.45	9	1	15.65	9	1	78.00
9	0	35.04	9	0	484.32	9	0	19.49	9	0	97.44
8	3	40.05	8	3	553.51	8	3	22.18	8	3	111.36
8	2	48.72	8	2	645.76	8	2	26.02	8	2	129.60
8	1	58.06	8	1	774.91	8	1	31.30	8	1	158.00
8	0	70.08	8	0	968.64	8	0	38.99	8	0	194.88

アタックタイム			サスティン/ディケイ/リリースタイム			アタックタイム			サスティン/ディケイ/リリースタイム		
レート		mSEC (0dB-96dB)	レート		mSEC (0dB-96dB)	レート		mSEC (10%-90%)	レート		mSEC (10%-90%)
7	3	80.09	7	3	1107.02	7	3	44.35	7	3	222.72
7	2	93.44	7	2	1291.52	7	2	52.03	7	2	259.20
7	1	112.13	7	1	1549.82	7	1	62.59	7	1	312.00
7	0	140.16	7	0	1937.28	7	0	77.95	7	0	388.76
6	3	160.18	6	3	2214.03	6	3	86.70	6	3	445.44
6	2	186.88	6	2	2583.04	6	2	104.06	6	2	518.40
6	1	224.25	6	1	3099.65	6	1	125.18	6	1	624.00
6	0	280.32	6	0	3874.55	6	0	155.90	6	0	779.52
5	3	320.37	5	3	4428.07	5	3	177.41	5	3	890.88
5	2	373.76	5	2	5166.08	5	2	208.13	5	2	1036.80
5	1	448.51	5	1	6199.30	5	1	250.37	5	1	1248.00
5	0	560.64	5	0	7749.12	5	0	311.81	5	0	1559.04
4	3	640.73	4	3	8856.14	4	3	354.82	4	3	1781.75
4	2	747.52	4	2	10332.15	4	2	416.25	4	2	2073.60
4	1	837.02	4	1	12398.59	4	1	500.74	4	1	2496.00
4	0	1121.28	4	0	15498.24	4	0	623.62	4	0	3118.08
3	3	1281.46	3	3	17712.27	3	3	709.63	3	3	3563.52
3	2	1495.04	3	2	20664.32	3	2	832.51	3	2	4147.20
3	1	1734.05	3	1	24797.19	3	1	1001.47	3	1	4982.00
3	0	2242.56	3	0	30996.48	3	0	1247.23	3	0	6236.16
2	3	2582.93	2	3	35424.55	2	3	1419.26	2	3	7127.04
2	2	2990.08	2	2	41328.64	2	2	1665.02	2	2	8234.40
2	1	3588.10	2	1	49534.37	2	1	2002.94	2	1	9984.00
2	0	4485.12	2	0	51992.96	2	0	2494.46	2	0	12472.32
1	3	5980.16	1	3	82657.28	1	3	3330.05	1	3	16588.80
1	2	5980.16	1	2	82657.26	1	2	3330.05	1	2	16588.80
1	1	8970.24	1	1	123985.92	1	1	4688.91	1	1	24914.84
1	0	8970.24	1	0	123985.92						

▼表Ⅰ-5-32　サスティンレートの各ビットと減衰量

ビット	7	6	5	4
減衰量(dB)	24	12	6	3

ただし，ビット7～4がすべて1の場合は93dB

● SSG型のエンベロープ生成

FM音源は，SSG(旧タイプのサウンドジェネレータで単純な波形を出力する)型のエンベロープ制御を行うことができます．波形のコードは，表I-5-33のとおりで，この値を内部レジスタのSSG-Typeに値を設定します．

なお，このときのアタックレートは通常，1FHである必要がありますが，＊印の波形の場合，表I-5-31に従って与えることができます．

SSGタイプの減衰時間は表I-5-34に従います．

SSGタイプのアタックレート，ディケイレートは，それぞれ，内部アドレスの50H～5EH，60H～6EHに設定します．

SSG型の波形はノコギリ波または三角波です．ディケイレート，サスティンレートの考え方が，立ち上りと立ち下りとでは図I-5-25のように異なるので，注意してください．

▼表I-5-33　SSG-TYPEのエンベロープを設定するレジスタ

レジスタ番号	7	6	5	4	3	2	1	0
90H～9EH					SSG-TYPE			

93H，97H，9BHは除く．

SSG-TYPE	エンベロープ波形
1　0　0　0＊	
1　0　0　1＊	
1　0　1　0	
1　0　1　1＊	
1　1　0　0	
1　1　0　1	
1　1　1　0	
1　1　1　1	

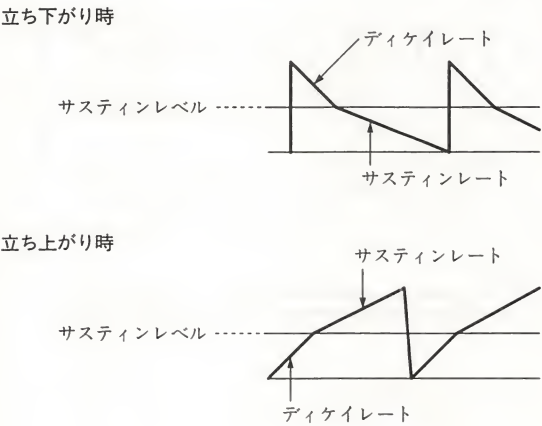
▼表 I-5-34 SSGタイプエンベロープ制御時の時間関係

ディケイタイム					
レート		mSEC (0dB-95dB)	レート		mSEC (0dB-95dB)
15	3	0.96	7	3	140.43
15	2	0.96	7	2	163.84
15	1	0.96	7	1	196.61
15	0	0.96	7	0	245.76
14	3	1.10	6	3	280.97
14	2	1.28	6	2	327.68
14	1	1.54	6	1	393.22
14	0	1.92	6	0	491.52
13	3	2.19	5	3	561.74
13	2	2.56	5	2	655.36
13	1	3.07	5	1	786.43
13	0	3.84	5	0	983.04
12	3	4.39	4	3	1123.47
12	2	5.12	4	2	1310.72
12	1	6.14	4	1	1572.86
12	0	7.68	4	0	1966.08
11	3	8.78	3	3	2246.95
11	2	10.24	3	2	2621.44
11	1	12.29	3	1	3145.73
11	0	15.36	3	0	3932.16
10	3	17.55	2	3	4493.90
10	2	20.48	2	2	5242.98
10	1	24.58	2	1	6291.46
10	0	30.72	2	0	7864.32
9	3	35.11	1	3	10485.76
9	2	40.96	1	2	10485.76
9	1	49.15	1	1	15728.64
9	0	61.44	1	0	15728.64
8	3	70.22			
8	2	81.92			
8	1	98.30			
8	0	122.88			

ディケイタイム					
レート		mSEC (10%-90%)	レート		mSEC (10%-90%)
15	3	0.38	7	3	55.44
15	2	0.38	7	2	65.04
15	1	0.38	7	1	77.52
15	0	0.38	7	0	97.68
14	3	0.42	6	3	110.88
14	2	0.50	6	2	130.08
14	1	0.58	6	1	155.04
14	0	0.76	6	0	195.38
13	3	0.85	5	3	221.76
13	2	1.02	5	2	260.16
13	1	1.21	5	1	310.08
13	0	1.52	5	0	390.72
12	3	1.73	4	3	443.52
12	2	2.03	4	2	520.32
12	1	2.42	4	1	620.16
12	0	3.05	4	0	781.44
11	3	3.46	3	3	887.04
11	2	3.07	3	2	1040.54
11	1	4.85	3	1	1240.32
11	0	6.11	3	0	1562.88
10	3	6.93	2	3	1774.08
10	2	8.13	2	2	2081.28
10	1	9.69	2	1	2480.64
10	0	12.21	2	0	3125.76
9	3	13.86	1	3	4162.56
9	2	16.26	1	2	4162.56
9	1	19.38	1	1	6251.52
9	0	24.42	1	0	6251.52
8	3	27.72			
8	2	32.52			
8	1	38.76			
8	0	48.84			

▼図 I-5-25 SSG 型波形のディケイレート

立ち下がり時

立ち上がり時

5.4.9　チャネル単位に設定する内部レジスタ

ここでは，内部レジスタのうちチャネル単位に設定を行うレジスタについて説明します．

●音程の設定

FM 音源の音程(ピッチデータ入力の発生周波数)は，各チャネルごとに内部レジスタ A0H～A6H(A3H は除く)で設定します(表 I-5-35)．

BLOCK はオクターブを表すもので，8 オクターブのなかから選びます．BLOCK 値(３ビット：１～７)は，８を除きオクターブ値(１～８)と同じ値を指定することになります．

F-Number は，オクターブ内での周波数のオフセットを表すもので，11ビットの周波数コードを設定します．

A の音と BLOCK の関係は表 I-5-36，音階と F-Number の関係は表 I-5-37のようになります．

図 I-5-26に音程設定の例を示します．

なお，音程の設定時には，最初に BLOCK と F-Number の上位３ビット(F-Number2)を書き込み，続いて下位８ビット(F-Number1)を書き込みます．この順序を間違うと正しい設定ができなくなります．

音程の設定は，チャネル４，５，６の場合も同様です．

チャネル３が，効果音モードまたは，音声合成モードの場合には，表 I-5-38のようにスロットごとに，周波数を設定することができます．

▼表 I-5-35　F-Number/Blockを設定するレジスタ

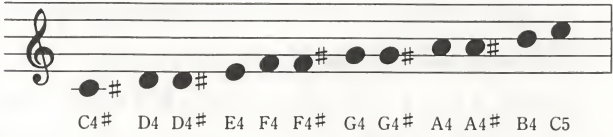
レジスタ番号	7	6	5	4	3	2	1	0	チャネル
A0H～A2H	F-Number 1								1～3(4～6)
A4H～A6H			BLOCK			F-Number 2			1～3(4～6)

▼表 I-5-36　Aの音の周波数とBLOCKの関係

Aの音	周波数	BLOCK	オクターブ
A8	7040.0	*	8
A7	3520.0	7	7
A6	1760.0	6	6
A5	880.0	5	5
A4	440.0	4	4
A3	220.0	3	3
A2	110.0	2	2
A1	55.0	1	1

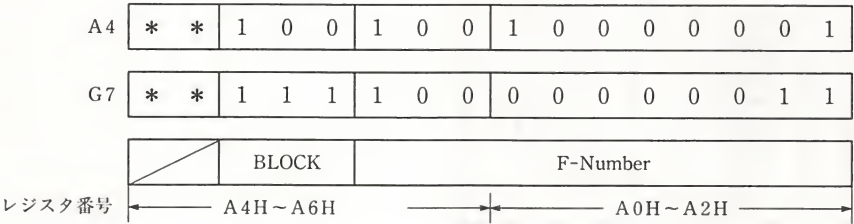
*の部分は，例えばBLOCK＝７，マルチプル＝２のように設定する．

▼表 I-5-37　音階とF-Numberの関係(BLOCK＝4の場合)

音階	周波数	F-Numberの設定値
C5	523.3	1371
B4	493.9	1294
A4#	466.2	1222
A4	440.0	1153
G4#	415.3	1088
G4	392.0	1027
F4#	370.0	969
F4	349.2	915
E4	329.6	864
D4#	311.1	815
D4	293.7	769
C4#	277.2	726

F-Numberの値は，異なるBLOCKの場合も同じになる．

▼表 I-5-38　チャネル3が特別設定の場合の周波数指定

レジスタ番号	7	6	5	4	3	2	1	0	チャネル	スロット
A2H	F-Number 1								3	4
A6H			BLOCK			F-Number 2			3	4
A8H	F-Number 1								3	3
A9H	F-Number 1								3	1
AAH	F-Number 1								3	2
ACH			BLOCK			F-Number 2			3	3
ADH			BLOCK			F-Number 2			3	1
AEH			BLOCK			F-Number 2			3	2

▼図 I-5-26　音程の設定例

●スロットの接続パターンの設定

スロット1のフィードバック(Self-Feedback)と，スロットの接続(Connection)のビット構成は表I-5-39のとおりです．

フィードバックとは，スロット1の出力を自己の入力に帰還させることです．

スロットの接続，つまりアルゴリズムの選択は，表I-5-17に示すアルゴリズムのパターン番号を設定します．

▼表I-5-39　Self-Feedback/Connectionの設定レジスタ

レジスタ番号	7	6	5	4	3	2	1	0	チャネル
B0H〜B2H			Self-Feedback			Connection			1〜3(4〜6)

Self-Feedback (bit5-3)　：フィードバックの帰還量を設定する．

設定値	0	1	2	3	4	5	6	7
帰還値	OFF	$\pi/16$	$\pi/8$	$\pi/4$	$\pi/2$	π	2π	4π

Connection (bit2-0)　：0〜7のアルゴリズムを選ぶ．

●周波数変調，振幅変調の設定

振幅変調をかけるとビブラートとなり，周波数変調をかけるとトレモロとなります．

これらの設定には，表I-5-40の振幅変調度(AMS)，位相変調度(PMS)を使用します．

位相変調とは，周波数変調のことをいいます．

▼表I-5-40　変調度などの設定レジスタ

レジスタ番号	7	6	5	4	3	2	1	0	チャネル
B4H〜B6H	L	R	AMS			PMS			1〜3(4〜6)

L, R (bit7, 6)　：音声ライン出力の有無を設定する．
0＝出力しない
1＝出力する

AMS (bit5, 4)　：振幅変調度を設定する．

AMS	0	1	2	3
変調度(dB)	0	1.4	5.9	11.8

PMS (bit2-0)　：位相変調度を設定する．

PMS	0	1	2	3	4	5	6	7
変調度(セント)	0	±3.4	±6.7	±10	±14	±20	±40	±80

●音の左右への出力

各チャネルは，左右いずれか一方，または両方に出力することができます．この選択には，変調度などを設定するレジスタのビット7とビット6を使います．ビット7が左(L)，ビット6が右(R)の音声ラインへの出力の有無を表し，設定値が1のとき出力され，0のとき出力されません．両方に出力するには，左右のいずれも1にすればよいわけです．

このような方法によっているので，左右の中央に定位させることはできますが，「右寄り」とか「左寄り」といった微妙な設定はできません．また，中央定位では片側だけのときに比べて合計レベルが倍になるため，そのチャネルだけ，音量が大きすぎることがあります．その場合は，トータルレベルの減衰量を増やす(例えば6dB＝半分とする)などして対応します．

5.5 LEDの制御

FMTOWNSの本体前面にあるLEDは，スピーカ(またはヘッドホン)の出力信号のレベルとボリューム設定レベルを表示する機能があります．

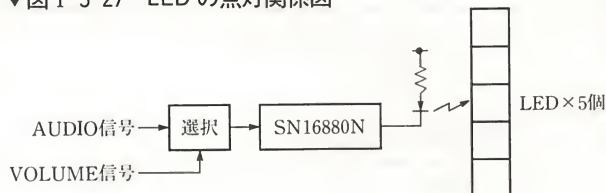
この節では，LEDのこの機能について解説します．

5.5.1 LEDを制御する2つの系統

LEDには，スピーカ(またはヘッドホン)の出力信号のレベルを刻々と表示する機能があります．

また，スピーカ(またはヘッドホン)のボリュームの設定レベルを表示する機能もあります．ボリュームの設定はUP/DOWNボタンで行いますが，このボタンを押すと数秒間ボリュームの設定されているレベルを表示します．

LEDの点灯関係のブロック構成は図I-5-27のとおりです．

▼図I-5-27　LEDの点灯関係図

5.5.2　LEDの点灯状態

出力信号レベルと5つのLEDの点灯関係は，表I-5-41のとおりで，信号の強さに応じて棒グラフ様に点灯します．

また，ボリューム設定値の表示も同様に行われます．なお，出力信号表示中にUP/DOWNボタンを押した場合には，数秒間はボリューム設定レベルを表示しますが，その後，出力信号レベルの表示にもどります．

▼表I-5-41　ローカル系のレベル表示

オーディオ信号の大きさ	LED0	LED1	LED2	LED3	LED4
小	1	1	1	1	1
↑	0	1	1	1	1
	0	0	1	1	1
中	0	0	0	1	1
	0	0	0	0	1
↓ 大	0	0	0	0	0

1　消燈
0　点燈

5.5.3 LED制御のレジスタ

LED制御には，オーディオレジスタ(表I-5-42)を使用します．

表示制御ビットはLED点灯を行うか否かを設定するものです．これを1にすると，点灯制御は行われなくなります．

ミュートビットは，LEDの点灯そのものを制御するものではありませんが，オーディオ出力を禁止するか否かを設定します．これを0にするとオーディオ出力はすべて行われなくなるので，LEDの点灯は行われません．リセット時はこのビットは0になっているので，オーディオ出力を行うには，1にしなければなりません．また，このビットはオーディオ出力を一時的に止めたい場合などに使用すると便利です．例えば，電源OFF時の「プツン」という音をなくすにはこの方法を使います．

▼表I-5-42　オーディオレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04ECH	オーディオレジスタ	R	LOFF	MUTE	不定					
		W			1	1	1	1	1	1

LOFF(bit7)　：オーディオレベルインジケータ使用の有無を設定する．
0＝使用する
1＝使用しない

MUTE(bit6)　：オーディオ出力の有無を設定する．
0＝出力しない
1＝出力する

5.6 FM音源,PCM音源のミュートについて

FMTOWNSには，FM音源，PCM音源に対してミュートをかけるための回路があります．ミュートの作動／解除は，FM･PCMミュートレジスタ(表I-5-43)を使用します．

▼表I-5-43　FM･PCMミュートレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04D5	FM･PCMミュートレジスタ	R	不　定						FM MUTE	PCM MUTE
		W	0	0	0	0	0	0		

PCM MUTE　：0のときPCM音源をミュートする．
FM MUTE　：0のときFM音源をミュートする．

第 6 章

CD-ROMドライブ

CD-ROMは，音楽用CDと同じメディアにコンピュータのデータを記録し，外部記憶装置として使用するものです．

CD-ROMの記憶容量はフォーマット時で540MBあります．パソコンの外部記憶装置としてこれまで使われてきた，フロッピィディスクやハードディスクに比べて，容量が格段に増えています．例えば，膨大な文字データ，画像データを有する百科事典の内容をCD-ROMに格納し，自在に検索するといったことも可能になります．さらに，CD-ROM中には，コンピュータのデータ(文字データ，画像データ，プログラムなど)だけでなく，音楽データ(CDに記憶するのと同じデータ)を収容し，再生することもできます．

この章では，CD-ROMのフォーマットと，読み書きに使われるCD-ROMドライブのハードウェアの仕組み，CD-ROMドライブを制御するための内部レジスタについて解説します．

なお，CD-ROMドライブは，ミュージックCDとCD-ROMにかかわらず同一なので，本文中の説明では，CDドライブと表現します．

6.1 CD-ROMのデータの格納形式

この節では，CD-ROMにどのようにデータが格納されているかについて説明します．ただし，CD-ROMのフォーマットはたいへん複雑であるため，本書では紙幅の関係から基本的な説明に限っています．

6.1.1 セクタの並び方

CD-ROMは片面だけを記録面として使用します．

CD-ROMに格納されているデータの最小単位をセクタといいます．1枚のCD-ROM中には約270000個のセクタがあります．

フロッピィディスクやハードディスクでは，セクタの構成は図I-6-1のように，同心円状になっており，外側のセクタは内側のセクタよりも物理的に長くなっています．その結果，回転速

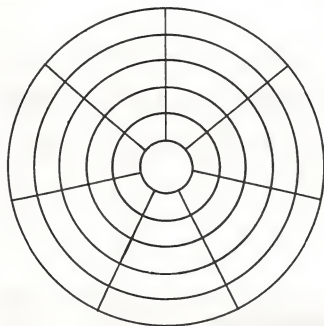
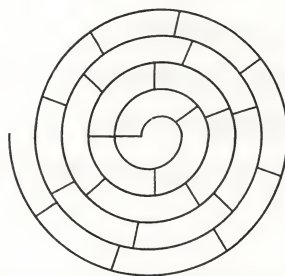
度一定で読み書きを行った場合，１つのセクタを読み取るのに要する時間は，外周と内周で同じになり，常に一定のスピードでデータを読み取ることができます．

いっぽう，CD-ROMのセクタは，CDとまったく同じで，図I-6-2のように内側から外側に向かって，螺旋状に並んでいます．また，セクタの長さはすべて同じになっています．これは，ディスク全体に一定の密度でデータが記録されているからです．

すべてのセクタの長さが同じであるため，１周当たりのセクタの数は，外周ほど多くなります．このため，CD-ROMでは，同じスピードでデータを読み取る（ヘッドに対するデータの線速度を一定にする）ために，回転速度を外周に行くほど低く，内周に行くほど高くしています．

このようなデータの格納形式は，CDで音楽を演奏する場合のように，連続してデータを読み取るときには都合がいいのですが，CD-ROMでは，ランダムアクセスが前提となる場合，若干の問題が生じます．

すなわち，ランダムアクセスでは，読み取りヘッドが外周と内周の間を頻繁に往復するので，その度にディスクの回転速度を変えなければならず，しかも速度が安定するまで，時間待ちが必要となり，アクセスに時間を要することになります．

▼図I-6-1　FD，HDのセクタの概念図

▼図I-6-2　CD-ROMのセクタの概念図

6.1.2 CD-ROMのフォーマット

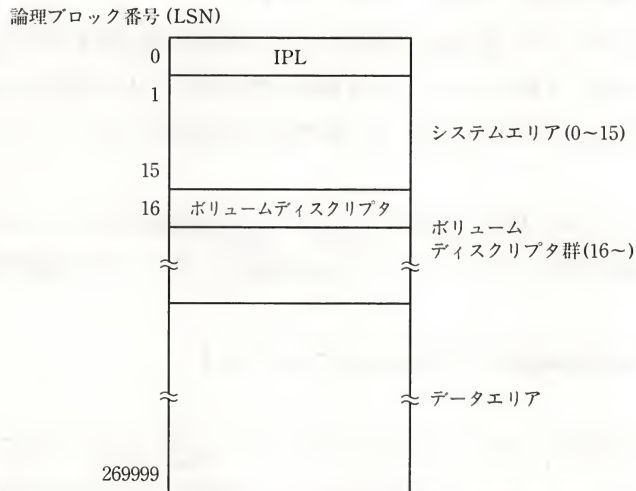
FMTOWNSのCD-ROMの論理的なデータ形式は，現在，最も標準的なISO9660の規格を採用しています．

ISO9660（情報交換用CD-ROMのボリュームとファイル構造）は，CD-ROMの標準化のため，記録形式とファイル構造について規定しています．

アドレスは時間値による物理アドレスの外に，2048バイト（または，それ以上の２のべき乗バイト）単位の論理セクタが規定されています．

また，CD-ROM空間は，図I-6-3のようにブロック分けがされています．０～15ブロックがシステム領域とされ，どのように利用するかはシステムの仕様に任されています．LSN16から

はデータ領域と呼ばれ，ボリュームディスクリプタ，パステーブル，ルートディレクトリなどのファイル管理領域に続いてディレクトリやファイルが並びます．

▼図Ｉ-6-3　CD-ROM空間

6.1.3　セクタのフォーマット

図Ｉ-6-4に，セクタ(データブロックともいう)のフォーマットを示します．

CD-ROMのデータブロックのフォーマットには，モード１とモード２があります．

モード２は，CDの音楽データ格納用に使用されていますが，CD-ROMでも使用できます．モード１はCD-ROM独自のフォーマットです．

モード２は，ブロックの先頭から，同期信号，ID，データの中身の順に格納されています．同期信号はCDドライブが連続するデータを正しく再生するため同期をとる目的で使われます．

IDは，セクタの位置を示すものです．CD-ROM内での論理的な位置を示します．CDでは，１つのセクタには75分の１秒(１フレーム)の音楽データが格納されており，セクタは，CDの先頭からの分，秒，セクタで指定します．CD-ROMでは，記憶されるデータが音楽データ以外の場合でも，セクタの指定にこの方法が使われています．

IDの１バイト目は分，２バイト目は秒，３バイト目はセクタを表します．４バイト目は，データブロックのフォーマットのモードを表します．

データ部は，2336バイトです．ここに，コンピュータのデータや音楽データが記憶されます．

モード１は，モード２にエラー訂正機能を付け加えたものです．

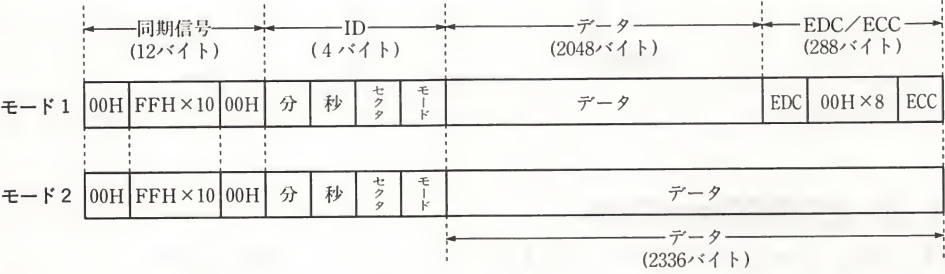
このためデータ部が2048バイトに減り，エラー検出と訂正のために288バイトが使われています．そのうちEDC(Error Detection Code)は４バイトで，エラーの検出に使用されます．ECC

(Error Correction Code)は276バイトあります．EDCの内容はCDコントローラでチェックされ，必要があれば，ECCを使った自動訂正やステータスによるエラー発生の通知を行います．

モード1は，間違いが許されないデータ(プログラムなど)の記録に使用し，モード2は，多少，間違いがあっても許される音楽データ，画像データの記録に使用するといった使い分けをすると，CD-ROMを効率よく使うことができます．

なお，CD中からセクタを読み出す段階で，エラー訂正処理が行われており，エラー発生率は10の9乗バイト当たり1個というレベルを実現しています．CDを演奏する上では，まったく問題のない状態になっています．このエラー処理は，CD-ROMのモード1，モード2でも行われています．

さらに，モード1では，EDCとECCによるエラー訂正機能により，エラー発生率は10の13乗バイト当たり1個となり，コンピュータの外部記憶として，十分な信頼性を得ています．

▼図Ⅰ-6-4　CD-ROMのデータブロックのフォーマット

6.2　CDドライブ制御の概要

この節では，CDドライブの制御の仕組みを解説します．

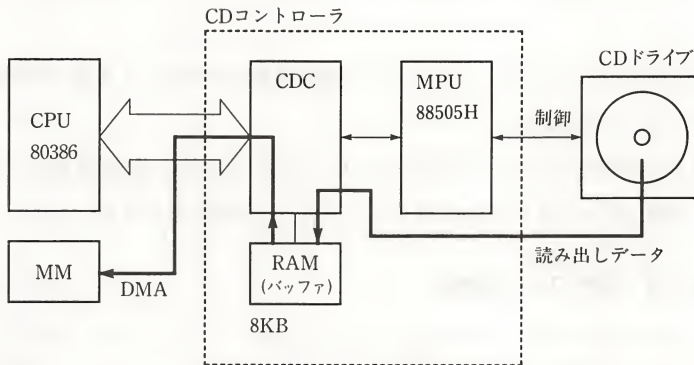
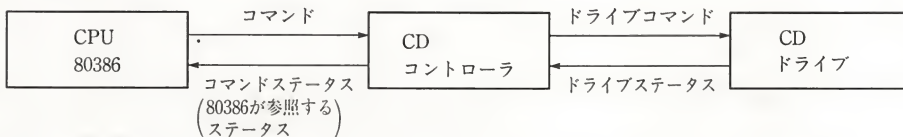
6.2.1　CDドライブ制御のメカニズム

図Ⅰ-6-5に，CDドライブ制御のブロック構成を示します．また，CDドライブの制御のコマンドの流れを図Ⅰ-6-6に示します．

CPUは，CDドライブに対して，直接，命令を出すことはできません．CPUがCDコントローラに対する命令を出し，CDコントローラがCDドライブに対するコマンドを出すというメカニズムになっています．

また，CDドライブの状態(ステータス)は，CDコントローラが参照できるようになっています．

CPUは，間接的にCDコントローラから，そのステータスを知ることができます．

▼図Ⅰ-6-5　CDドライブのブロック図

▼図Ⅰ-6-6　CDドライブ制御の流れ

RAMバッファは，CD-ROMから読み出したデータを一時的に格納するメモリです．

CDドライブは，CDコントローラを経由してバスに接続されています．

CDコントーラには，富士通のワンチップマイコンMB88505(以下SUB　MPUと呼ぶ)とCDCが搭載されています．

SUB MPUは，おもにCDドライブに対する直接の制御を行っています．CDCはCPU，SUB MPUの間のデータ転送，CDから読み出したデータのメモリへの転送(CPUまたはDMAが行う)などに使用されます．このようなデータ転送には，CDC内部のレジスタが使用されます．このレジスタについての説明は，「6.3　CDドライブ関係のレジスタ」を参照してください．

6.2.2　CDドライブ制御の流れ

CDドライブの制御の流れを図Ⅰ-6-7に示します．

●初期化

①CPUからCDCに対して，リセットを指示する信号を送出します．マスタコントロールレジスタ(表Ⅰ-6-1)を使用します．

②CDCはCDドライブに対して，リセット指示を出します．

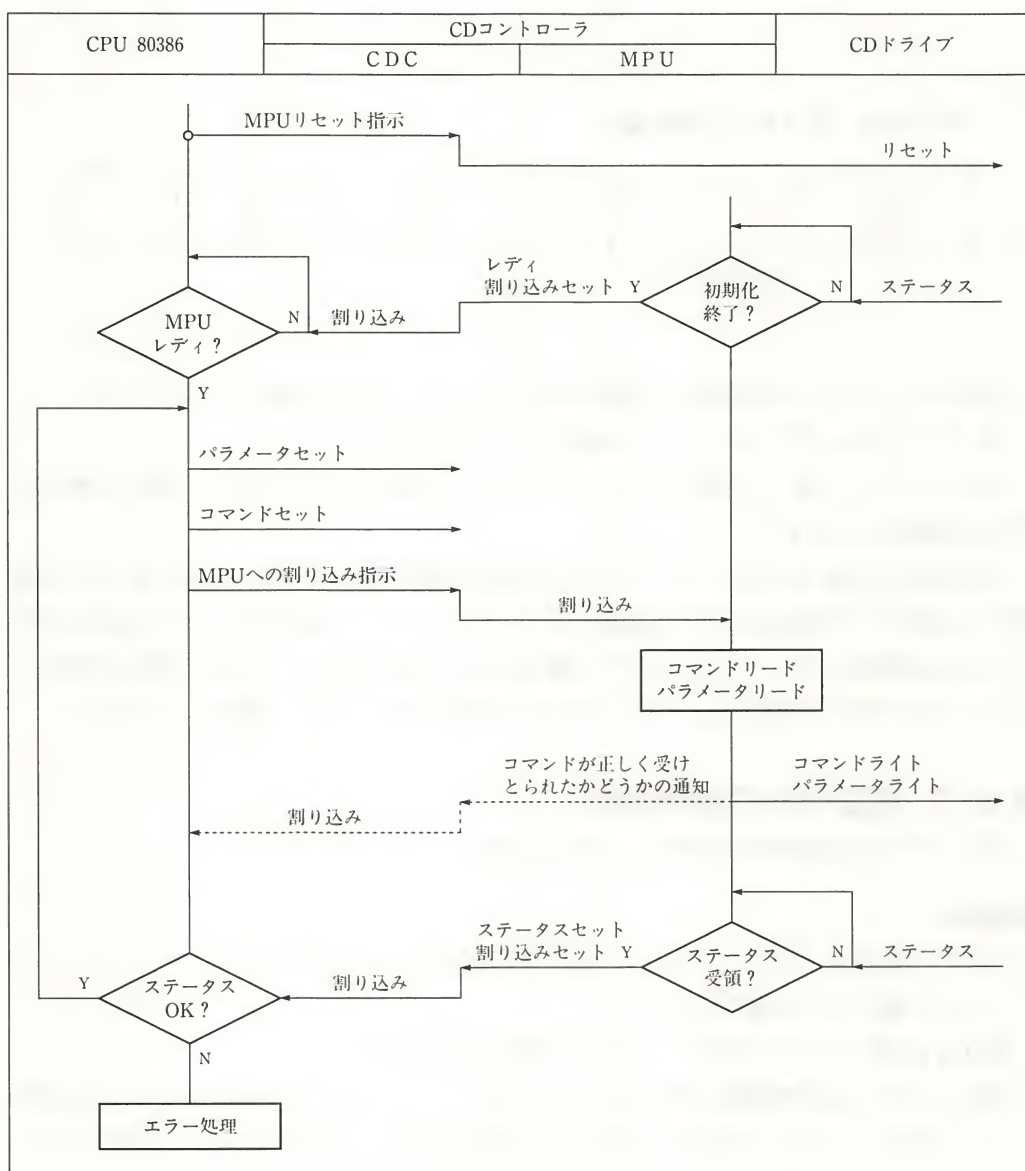
③CDドライブは初期化動作(例えば，ドライブ内ステータスなどのクリア)をし，それが終わるとCDコントローラのMPUに対して，初期化が終了した旨のステータスを送ります．

④MPUは，CDCに対してCDドライブの初期化動作が完了したことを通知します．

⑤CDCはCPUに対して割り込みをかけます．CPUは次のコマンドを送ることができるようになります．

CPUへの通知は割り込みによって行うので初期化動作が終了するまでの間にCPUは別な処理を行うことができます．

なお，CPUへの割り込みは，マスタコントロールレジスタのSMIMを0にするとマスクされるので，割り込みを使う場合には1にしておく必要があります．

▼図Ⅰ-6-7　CDドライブ動作フロー概念図

●一般のコマンド

CDドライブを制御する，CPUからのコマンドは次のような流れで処理されます．

①CPUは，CDコントローラに対して，コマンドとパラメータを送出します．コマンドレジスタ(表I-6-3)とパラメータレジスタ(表I-6-4)を使用します．
②CDCはCDコントローラのSUB MPUに対して，割り込みをかけます．SUB MPUはコマンドとパラメータを読み出し，CDドライブに必要な命令を送ります．
③CDドライブはその命令を実行し，実行が終わるとSUB MPUに終了のステータスを送ります．
④SUB MPUはCDCに対して，CDの動作が終了したことを通知します．
⑤CDCは，CPUに割り込みをかけます．CPUは，新しいコマンドを送るように動作します．CPUへの割り込みは，マスタコントロールレジスタのSMIMを0にするとマスクされるので，割り込みを使うときには1にしておく必要があります．

このとき，CPUが続けてコマンドを送るようにプログラムしておくことにより，続けてCDドライブを制御することができます．

CPUからコマンドを送ってから，コマンドの終了を通知する割り込みがかかるまでの間は，CPUは別の動作を行うことができます．

また，CPUからCDCへ送るコマンドレジスタのIRQとSTAUSを1にしておくと，SUB MPUがCDドライブにコマンドを送った時点で，CPUに割り込みがかかるようになります．これは，そのコマンドが正しく受けとられたかどうかを，ステータスを参照して，すぐチェックしたいときに使われます．

6.3　CDドライブ関係のレジスタ

ここでは，CDドライブ制御に関係するCDC内部のレジスタを示します．しかしCDCに対するコマンドやCDCから返されるステータスやパラメータのフォーマットについては，公開されていません．CD-ROMを使ったプログラムを作成する場合には，CD-ROM BIOSを使用してください．

CDCの内部には，書き込み専用と読み出し専用のレジスタがあります．

・書き込み専用のレジスタ

マスタコントロールレジスタ…割り込みの設定
コマンドレジスタ………………コマンドの設定
パラメータレジスタ……………パラメータの設定
転送制御レジスタ………………データ転送モードの設定

・読み出し専用レジスタ

マスタステータスレジスタ……………割り込み関係のステータスを読み出す

ステータスレジスタ……………………コマンドステータスを読み出す

データレジスタ…………………………ソフトウェア転送の際，データを読み出す

CDサブコードステータスレジスタ…サブコード読み出しの際にステータスを与える

CDサブコードデータレジスタ………サブコードのデータ値を読み出す

●マスタコントロールレジスタ

マスタコントロールレジスタ(表Ⅰ-6-1)は，割り込み要求のマスク／許可と，割り込み対応ハンドラ(割り込み処理のプログラム)で割り込み要求を解除するクリアビットを持っています．

また，初期化時にSUB MPUをリセットするためのSRSTも用意されています．

●マスタステータスレジスタ

マスタステータスレジスタ(表Ⅰ-6-2)は，割り込み対応ハンドラで，割り込み要因を調べるときに参照します．

●コマンドレジスタ

コマンドレジスタ(表Ⅰ-6-3)は，CDに対するコマンドバイトを書き込むのに使用します．

TYPEはコマンドの性質を表すもので，PLAY系は，音楽演奏，データリードなど，CDドライブを実際に動かすコマンドのことを，STATE制御系は動作モードの変更など設定値のみを変えるコマンドを示します．

▼表Ⅰ-6-1　マスタコントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C0H	マスタコントロールレジスタ	W	SMIC	DEIC	0	0	0	SRST	SMIM	DEIM

SMIC(bit7)　：SUB MPU IRQ CLEAR.
　1＝SUB MPUからの割り込み要求をクリアする

DEIC(bit6)　：DMA END IRQ CLEAR.
　1＝DMA転送終了割り込み要求をクリアする

SRST(bit2)　：SUB MPU RESET.
　1＝SUB MPUをリセットする

SMIM(bit1)　：SUB MPU IRQ MASK.
　0＝SUB MPUからの割り込み要求を禁止する
　1＝SUB MPUからの割り込み要求を許可する

DEIM(bit0)　：DMA END IRQ ENABLE MASK.
　0＝DMA転送終了割り込みを禁止する
　1＝DMA転送終了割り込みを許可する

bit3，bit2の各ビットは，転送終了時にハードウェアによりリセットされる．
bit6は，DMA転送終了時にセットされる．

▼表 I-6-2　マスタステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C0H	マスタステータスレジスタ	R	SIRQ	DEI	STSF	DTSF	不定		SRQ	DRY

SIRQ (bit7)　: SUB MPU IRQ.
　0＝SUB MPUからの割り込み要求がない
　1＝SUB MPUからの割り込み要求がある

DEI (bit6)　: DMA END IRQ.
　0＝DMA転送終了割り込み要求がない
　1＝DMA転送終了割り込み要求がある

STSF (bit5)　: SOFT TRANS.
　0＝ソフト転送終了
　1＝ソフト転送中

DTSF (bit4)　: DATA TRANS.
　0＝DMA非転送
　1＝DMA転送中

SRQ (bit1)　: STATUS READ REQUEST.
　0＝ステータスリード要求がない
　1＝SUB MPUがコマンド実行後にステータスリード要求がある

DRY (bit0)　: SUB MPU READY.
　0＝SUB MPUがコマンド受付不可能状態
　1＝SUB MPUがコマンド受付可能状態

bit7, bit1, bit0の各ビットは，SUB MPUによりセットされる．
bit6は，DMA転送終了時にセットされる．

▼表 I-6-3　コマンドレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C2H	コマンドレジスタ	W	TYPE	IRQ	STATUS	COMMAND CODE				

CPUからSUB MPUにコマンドを送るためのレジスタ．

TYPE (bit7)　: コマンドのタイプを示す
　0＝PLAY系コマンド
　1＝STATE制御系コマンド

IRQ (bit6)　: コマンドステータス要求時のIRQ制御
　0＝IRQオフ
　1＝IRQオン

STATUS (bit5)　: コマンドステータスの要求制御
　0＝要求しない
　1＝要求する

COMMAND CODE (bit4-0)　: コマンドコード

●パラメータレジスタ

パラメータレジスタ(表I-6-4)は，CPUからSUB MPUに送るコマンドのパラメータを書き込むのに使用します．このレジスタは8段階のFIFO(ファーストインファーストアウト)構造となっているので，未定義の部分も続けて8個分書き込まなければなりません．

▼表I-6-4　パラメータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C4H	パラメータレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0

CPUからコマンドといっしょに動作モードを指定するレジスタ．
同一アドレスに8バイトのFIFOを構成しているので，同一アドレスに8回必ず書き込みを行うこと．

●ステータスレジスタ

ステータスレジスタ(表I-6-5)には，SUB MPUからのコマンドに対応するステータスが格納されています．CPUからの参照に使用します．このレジスタも4段階のFIFO構造となっているので，読み出し時には予約済の部分まで，続けて4回読み込まなければなりません．

▼表I-6-5　ステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C2H	ステータスレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0

SUB MPUからの実行終了後のステータス情報が格納される．
同一アドレスに4バイトのFIFOを構成しているので，同一アドレスから4回必ず読み出しを行うこと．

●転送制御レジスタ

転送制御レジスタ(表I-6-6)は，CD-ROMから読み出したデータをDMA転送するか，プログラムによりCPUから読み出して転送するかの設定に使用します．CPUから読み出す際には，DTSを0にし，STSを1にすることによって対応できます．DMAから読み出す場合は，DTSを1にします．

▼表 I-6-6　転送制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C6H	転送制御レジスタ	W	0	0	0	DTS	STS	0	0	0

DTS(bit4)　: DMA TRANSFER MODE.
0 =DMA データ転送モードでない
1 =DMA データ転送モードである

STS(bit3)　: SOFTWARE TRANSFER START.
1 =CPU の DR 読み出しでデータ転送が行える

●データレジスタ

データレジスタ(表 I-6-7)は，CD-ROM から読み出したデータの読み出し用の「窓」として働きます.

▼表 I-6-7　データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C4H	データレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0

ソフトウェア転送モードのときの読み出しのデータレジスタ.

● CD サブコードステータスレジスタ

CD サブコードステータスレジスタ(表 I-6-8)は，サブコードを参照するときに，サブコードが有効であるかどうかを調べます.

▼表 I-6-8　CD サブコードステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04CCH	CDサブコードステータスレジスタ	R	不　定						OVER RUN	SUBC DATR

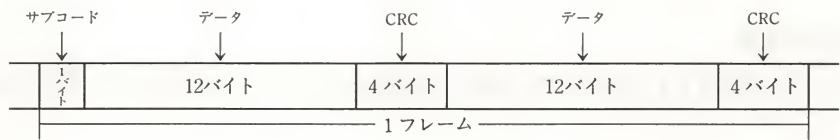
CD のサブコードを読み取るときのステータスレジスタ.
OVER-RUN(bit1)　: サブコードデータがデータレジスタにあるとき，新たなサブコードが入力されて前のデータが失われたことを示す.
SUBC-DAT-R(bit0) : サブコードデータ(P～W)がデータレジスタに入ったことを示す.

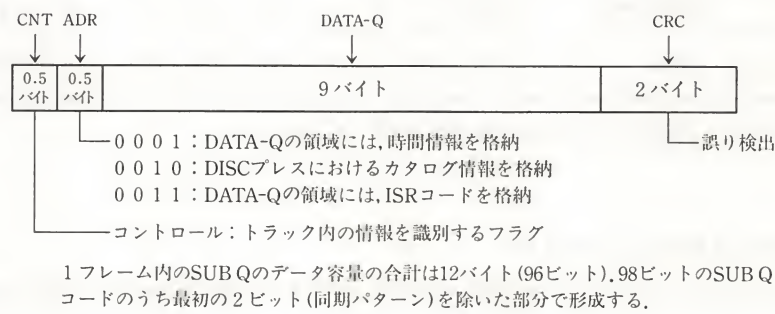
サブコードとは，CD のデータフレーム(図 I-6-8)ごとに付けられる 1 バイトの補助データのことで，各ビットは上位から P，Q，R～W ビットと名前が付けられています. 1 セクタは98個のデータフレームで構成されているので， 1 セクタ当たりのサブコートは合計98バイトと

なります．したがってP，Q，R～Wの各ビットの合計はそれぞれ98ビットです．これをそれぞれSUBPデータ，SUBQデータなどといいます．図I-6-9はサブコード中のQビットを，1セクタ分集めてみた場合(SUBQデータ)のフォーマットを示したものです．

普通，SUBQデータは，時間情報を形成します．

SUBR～SUBWは，一般に，CD再生中に表示する静示画のグラフィックデータを収容するのに用いられます．

▼図I-6-8　サブコードの位置

▼図I-6-9　CD-ROMのSUBQデータのフォーマット

1フレーム内のSUB Qのデータ容量の合計は12バイト(96ビット)．98ビットのSUB Qコードのうち最初の2ビット(同期パターン)を除いた部分で形成する．

●CDサブコードデータレジスタ

CDサブコードデータレジスタ(表I-6-9)は，サブコードを読み取るためのレジスタです．

サブコードデータは，OVER-RUNビットが0，SUBC-DAT-Rビットが1であるとき有効で，オバーラン発生(OVER RUNビット＝1)時には，データの取り逃がしがあったことを示します．オーバーランとは，CPUによってCDサブコードデータレジスタが読まれないうちに次のデータが入ってしまうことをいいます．

▼表I-6-9　CDサブコードデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04CDH	CDサブコードデータレジスタ	R	SUBC P-DATA	SUBC Q-DATA	SUBC R-DATA	SUBC S-DATA	SUBC T-DATA	SUBC U-DATA	SUBC V-DATA	SUBC W-DATA

CDのサブコードを読み取るときのデータレジスタ．

第7章

各種のデバイス

FMTOWNSは各種の入出力デバイスをサポートしています．

この章では，キーボード，TOWNSパッド，TOWNSマウス，プリンタ，フロッピィディスク，ハードディスク，RS-232Cインタフェースなどのデバイスについて解説します．

7.1 キーボード

FMTOWNSでは，キーボードはオプションとなっていますが，多数の文字を入力する際には，不可欠のデバイスです．この節では，キーボードとキーボードインタフェースの仕組みと働きについて解説します．

7.1.1 キーボードインタフェース概要

キーボードの仕様を，表I-7-1に示します．

▼表I-7-1　キーボードインタフェースの仕様

項　目	仕　　様
インタフェース	シリアル ・調歩 ・９６００bps ・８ビット＋偶数パリティ＋１ストップビット ・キーボードから本体への一方向転送 （特殊なキーボードについては，双方向も可能）
コントローラ	８０４２
キースキャナ	キーボード側に搭載（JIS／親指シフト） 使用プロセッサ　：８０４９

キーボードにはJIS配列のタイプ（図I-7-1）と親指シフトタイプ（図I-7-2）があり，それぞれにテンキーのないタイプとあるタイプがあります．ユーザーは，好みに応じて選択すること

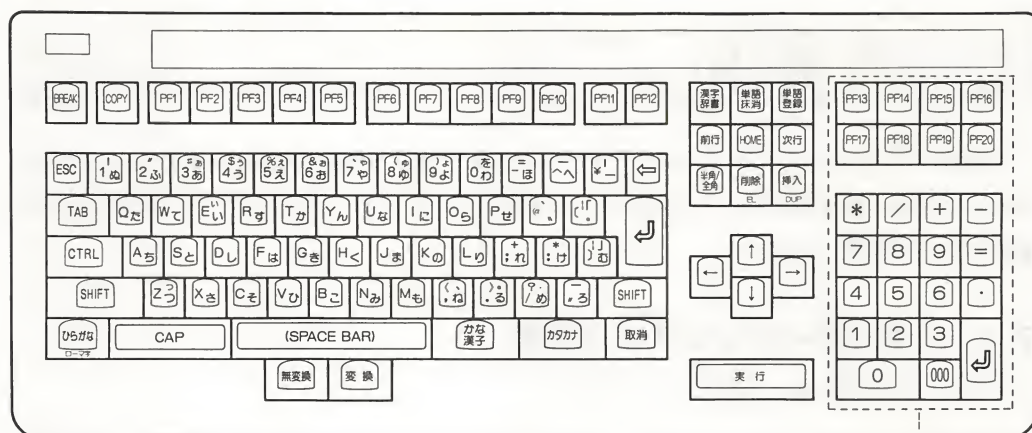
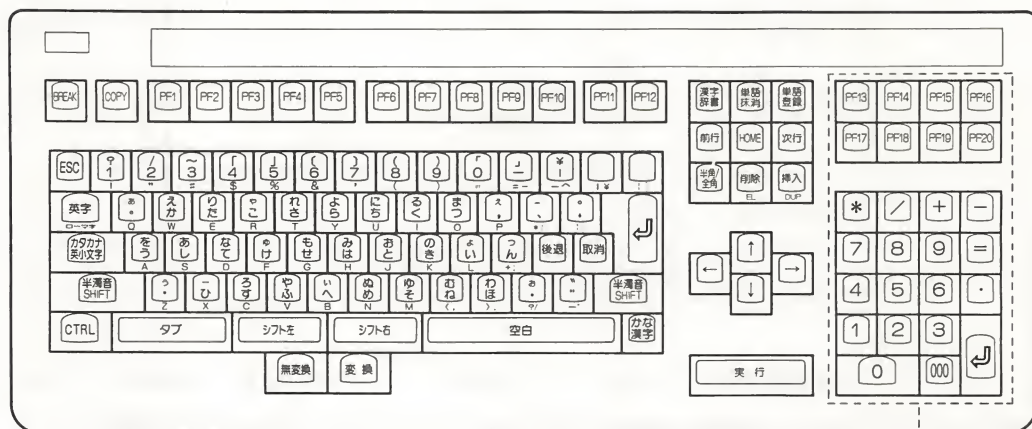
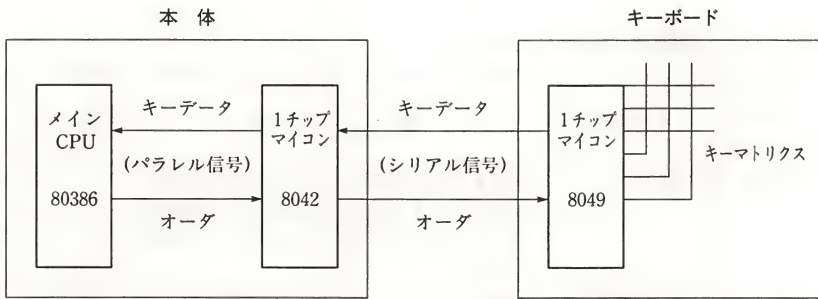
ができます.

キーボードは，マウスが接続できないことを除けば基本的にFMRシリーズのものと同じです.

キーボードインタフェースは，本体とキーボードを接続するためのシリアルインタフェースです．通常，キーボードから本体へデータを送るのみの一方通行ですが，特殊なデバイスを使用する場合には，双方向も可能です.

キーボード内部にはキーボード制御用のマイコンとしてインテル8049が搭載され，押されたキーの番号を検出して本体に送る役目を果たしています.

本体側(キーボードインタフェース)には，別のマイコンのインテル8042がキーボード専用として搭載され，キーボードからきたシリアル信号をパラレル信号に変換して，CPUに連絡する役目を果たしています(図I-7-3).

▼図I-7-1 JISキーボードの外観

▼図I-7-2 親指シフトキーボードの外観

▼図Ｉ-7-3　キーボード制御回路系

7.1.2 キーボード制御のレジスタ

ユーザーが作成したプログラムなどで，押されたキーを検出するなどの動作をさせるには，キーボードインタフェース(8042)に対してコマンドを送出し，その後，キーボードから8042に返されるデータを読み取るという方法を用います．

このような制御にかかわるレジスタが６つあります．

●コマンドレジスタ

8042へのコマンドの送出は，コマンドレジスタ(表Ｉ-7-2)に値を書き込むことによって行います．

コマンドのフォーマットには２種類あります．共通オーダは，キーボード用に使用されます．デバイスオーダは，キーボード以外の各種デバイスに対してコマンドを送る場合に使います．

▼表Ｉ-7-2　コマンドレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0602H	コマンドレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0

共通オーダのフォーマット

デバイスオーダのフォーマット

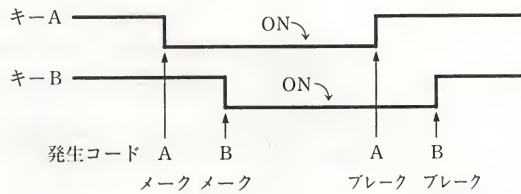
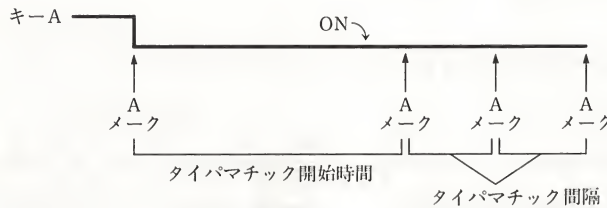
共通オーダの場合の設定値の意味を表I-7-3に示します.

▼表I-7-3 共通オーダのフォーマット

コマンドレジスタ設定値	オーダ名	機能	対象
10100001	RESET	このオーダを受けとるとデフォルト状態となる. データの送信中, 他のオーダの受信中など, いかなる場合でもこのオーダが優先する.	
10100100	同時打鍵監視モードオン	親指シフトキーボードの同時打鍵を有効とする.	本体側1チップマイコン用
10100101	同時打鍵監視モードオフ	親指シフトキーボードの同時打鍵を無効とする. デフォルトは無効.	〃
10101001 10101010 10101011 10101100 10101101 10101110	タイパマチック時間	タイパマチック開始時間を400msとする. 〃 500msとする. 〃 300msとする. タイパマチック周期を50msとする. 〃 30msとする. 〃 20msとする. *	デフォルトは400ms-30ms. 本体側1チップマイコン用
10110000	カーソル斜め移動有効	カーソルの斜め移動(2つのコードを交互に送信する)を有効とする(デフォルトは有効).	キーボード側1チップマイコン用
10110001	カーソル斜め移動無効	カーソルの斜め移動を無効とする.	〃
10110010	NMI ACK	メインCPUのNMIハンドラは, キーボードからのNMIであることを検出したら, 速やかにこのオーダを送らなければならない. それまでは本体側マイコンは処理を再開しない. また, NMIのACK以外でこのオーダを送ってはならない.	本体側1チップマイコン用
10110011 ∫ 10111111		拡張用予約済	

*のタイパマチック周期20msはマウスデータと交互に送信している場合などでは遅れることがある.

表Ｉ-7-3中のタイパマチックとは，キーを押し続けた場合，オートリピート動作により，連続してキーデータが送出されることをいいます．キーデータ送出のタイミングを図Ｉ-7-4に，オートリピート動作時のタイパマチック開始時間とタイパマチック周期の意味を図Ｉ-7-5に示します．メークというのは，キーが押されたときであり，キーが離されたときをブレークといいます．

▼図Ｉ-7-4　キーデータ送出のタイミング

▼図Ｉ-7-5　オートリピート動作

●キーボードデータレジスタ

キーボードから本体に送られてきたデータは，キーボードデータレジスタ(表Ｉ-7-4)から読み出します．キーボードインタフェースから返されるデータのフォーマットはこのようになっています．

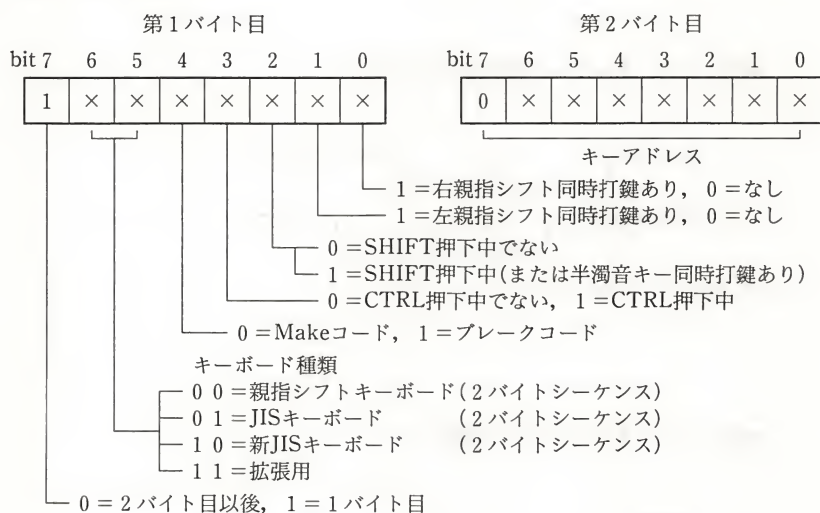
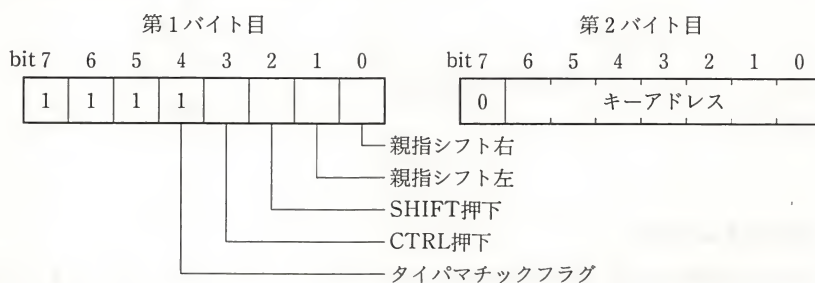
フォーマットには，４つの形式があります．キーデータは，通常のキー入力を読み出します．タイパマチックはキーリピートの場合に使います．

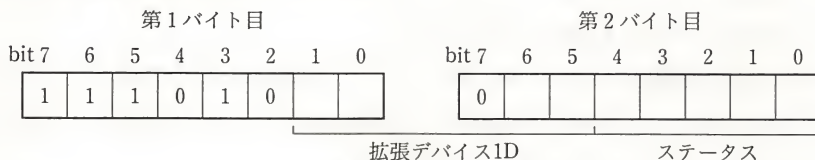
また，デバイス属性情報データは，キーボードとマウス以外のデバイスをキーボードインタフェースにつないだ場合，ステータスを知るときに使用します．また，拡張データは，任意のバイト数のデータを386CPUに引き渡すのに使用します．

なお，キーのアドレスは，図Ｉ-7-6と図Ｉ-7-7のようになっています．

▼表Ｉ-7-4　キーボードデータレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0600H	キーボードデータレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0

キーデータのフォーマット

タイパマチックデータのフォーマット

タイパマチックデータではキーボードの種類は，直前の種類と同じと判断する．
キーボードから本体に送出されるデータも基本的に上記と同じ形式に従う．

デバイス属性情報データのフォーマット

このデータ形式は，キーボード以外のデバイスがメインCPUにステータスを通知するために存在する．

拡張データのフォーマット

拡張データは，キーボード以外のデバイスをキーボードインタフェースを通じて接続するためにある．本体側１チップCPU(8042)は，何の加工もせずにこれをメインCPUに通知する．
CNTは２バイト目以降のデータ(２バイト目を含む)のバイト数を示し，０ならば最大32バイトとする．

▼図 I-7-6　JIS キーボードスキャンコード一覧

▼図 I-7-7　親指シフトキーボードスキャンコード一覧

●ステータスレジスタ

8042のステータスを調べるためのレジスタがステータスレジスタ(表I-7-5)です．コマンドを送出するときに，このレジスタを読み出してIBFが0であることを事前に確認しておく必要があります．このビットが1の場合には，書き込んだコマンドは無視されます．

また，OBFは8042から80386CPUに送るデータがあるとき1になるので，キーデータレジスタの値を読み出す際には，事前に1であることを確認します．

▼表I-7-5 ステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0602H	ステータスレジスタ	R	ST7	ST6	ST5	ST4	F1	F0	IBF	OBF

ST7-4(bit7-4) ：ユーザー定義可能なステータスビット(内容未定)．

F1(bit3) ：不定．

F0(bit2) ：汎用フラグ(8042の内部プログラムによってセットされる)．

IBF(bit1) ：8042の入力バッファの状態を示す．
　0＝データなし(データ書き込み可能)
　1＝データあり(データ書き込み禁止)
マスタCPU(80386)が8042の入力バッファにデータをライトすると，1になり，8042がそのデータをアキュムレータに移すと0になる．

OBF(bit0) ：8042の出力バッファの状態を示す．
　0＝データなし
　1＝データあり
8042が出力バッファにデータをロードすると，1になり，マスタCPU(80386)出力バッファにデータをリードすると0になる．

●8042データレジスタ

8042データレジスタ(表I-7-6)は，コマンドの第2バイトからの内容(パラメータ)が必要な場合，それらを書き込むのにも使われます．その場合，先に8042データレジスタに書き込みを行い，最後にコマンド第1バイトをコマンドレジスタに書きます．

▼表I-7-6 8042データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0600H	8042データレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0

●割り込み要因フラグレジスタ

割り込み要因フラグレジスタ(表I-7-7)は，割り込みの要因を調べることができます．

キーボードからの割り込みの有無を示すフラグ(KBINT)とキーボードインタフェースからの強制割り込みの有無を示すフラグ(NMI)の2つのビットがあります．

▼表I-7-7　割り込み要因フラグレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0604H	割り込み要因フラグレジスタ	R	不定						NMI	KBINT

NMI(bit1)　：キーボードインタフェースからのNMI割り込みの有無を示す．
0＝割り込みなし
1＝割り込みあり

KBINT(bit0)　：キーボードからの割り込み有無を示す．
0＝割り込みなし
1＝割り込みあり

●割り込み制御レジスタ

割り込み制御レジスタ(表I-7-8)は，割り込みを禁止するか許可するかのフラグ(KBMSK)があるのみです．

▼表I-7-8　割り込み制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0604H	割り込み制御レジスタ	W	0	0	0	0	0	0	0	KBMSK

KBMSK(bit0)　：キーボード割り込みの制御．
0＝割り込み禁止
1＝割り込み許可

7.2 TOWNSパッド

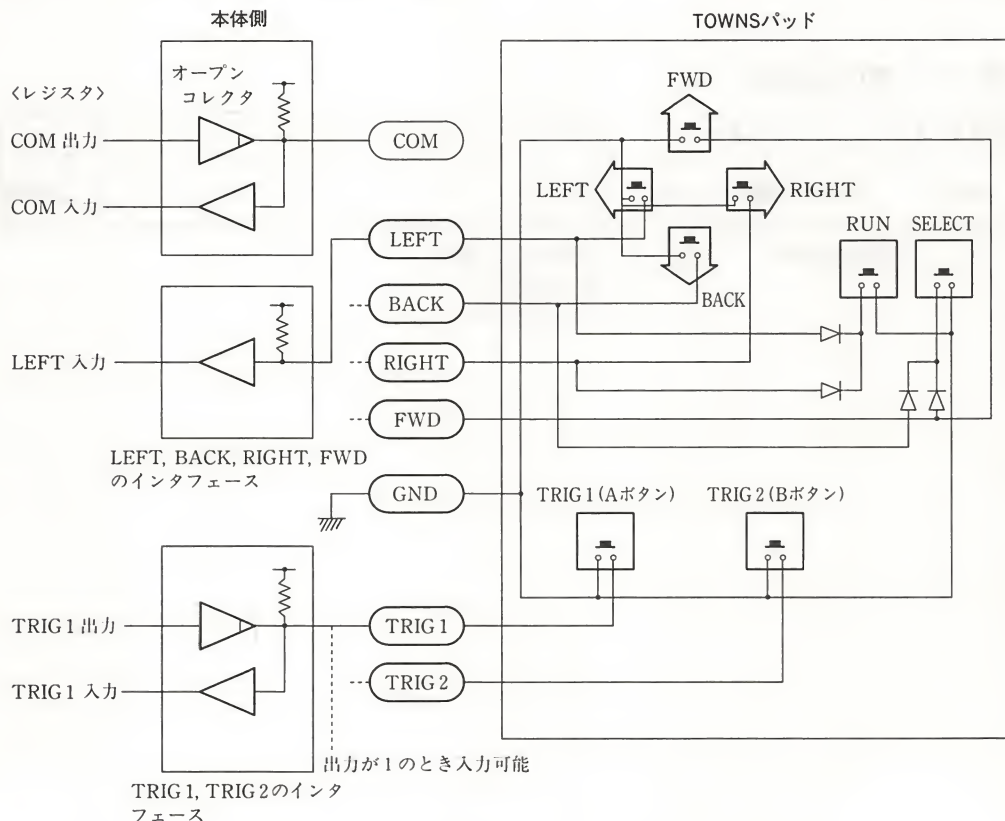
この節では，TOWNSパッドとインタフェースのハードウェアの仕組みと働きについて解説します．

7.2.1 TOWNSパッドインタフェース概要

本体とTOWNSパッドのインタフェースは，TOWNSパッドの上下左右の移動ボタンとAボタン，Bボタン，SELECTボタン，RUNボタンの押下状態が本体に入力される形態になっています(図I-7-8)．

TOWNSパッドの内部スイッチからの信号線は，すべて本体のGNDに接続されています．したがって，そのままで各スイッチが使用可能になっています．もし，MSX仕様のパッドと互換性をとる場合は，COM出力を0にします．このとき，COM入力も0となり，各スイッチの入力が可能になります．

なお，Aボタン，Bボタンの押下による信号を読み取るには，TRIG系の出力を1にしておく必要があります．

▼図I-7-8　TOWNSパッドと本体の信号関係

7.2.2 TOWNSパッドのレジスタ

TOWNSパッドは2系統同時に使用できるように，レジスタもそれぞれの系統に対応するように設計されています．

Aボタン，Bボタン，上下左右移動ボタン，RUNボタン，SELECTボタンの押下状態は，パッド1，2入力レジスタ(表I-7-9)から，読み取ることができます．

COM出力とTRIG出力の制御には，パッド出力レジスタ(表I-7-10)を使います．パッド1，2入力レジスタから値を読み出す前には，COM入力を読み出す場合には対応するフラグを0に，TRIG入力を読み出す場合には対応するフラグを1にしておく必要があります．

▼表I-7-9　パッド1，2入力レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04D0H	パッド1入力レジスタ	R	／	COM	TRIG2	TRIG1	RIGHT	LEFT	BACK	FWD
04D2H	パッド2入力レジスタ	R	／	COM	TRIG2	TRIG1	RIGHT	LEFT	BACK	FWD

TOWNSパッドの各ボタンの押下げ状態を示す．

COM(bit6)　：COM入力を示す．
0＝COM入力がない
1＝COM入力がある

TRIG1，2(bit5，4)　：Aボタン(TRIG1)，Bボタン(TRIG2)の状態を示す．
0＝ボタンが押された
1＝ボタンが押されていない

RIGHT	LEFT	BACK	FWD	意　味
0	1	1	1	方向キー(右)が押された
1	0	1	1	方向キー(左)が押された
1	1	0	1	方向キー(下)が押された
1	1	1	0	方向キー(上)が押された
0	0	1	1	RUNボタンが押された
1	1	0	0	SELECTボタンが押された

▼表I-7-10　パッド出力レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04D6H	パッド出力レジスタ	W	0	0	JOY2 COM	JOY1 COM	JOY2 TRIG2	JOY2 TRIG1	JOY1 TRIG2	JOY1 TRIG1

TOWNSパッドへの，制御情報レジスタである．

7.3 TOWNSマウス

この節では，TOWNSマウスとインタフェースの仕組みと働きについて解説します．

7.3.1 TOWNSマウスインタフェース概要

TOWNSマウスのインタフェースのコネクタは，TOWNSパッドと共用になっています．

図I-7-9にTOWNSマウスとインタフェースのブロック図を示します．

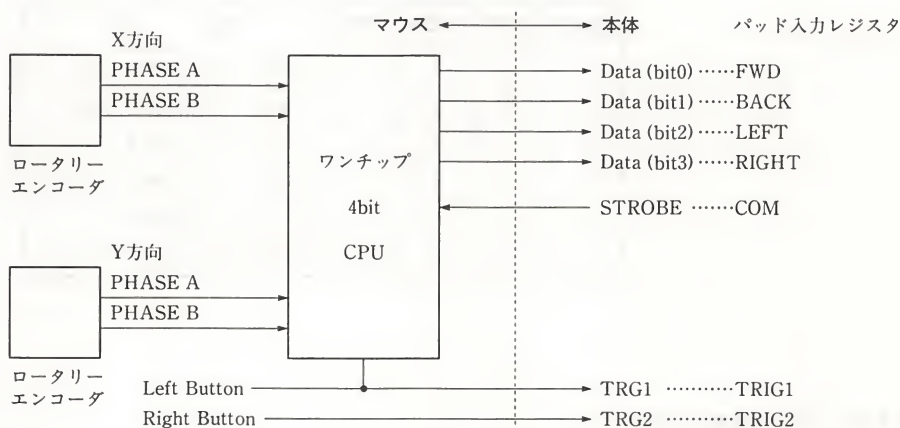
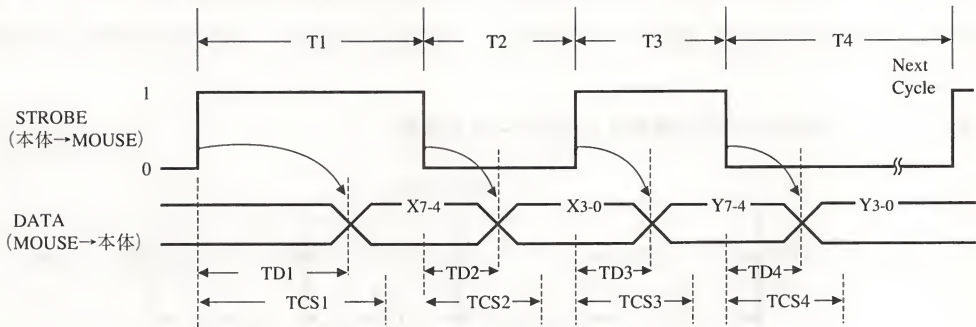
移動量の読み取りの制御は，次のように行います．

マウスの移動データは，4ビット(1ニブル)ずつ転送されます．COM出力を0→1，または1→0に変化させると，マウスは1ニブルのデータを送出します．この動作を4回繰り返すことにより，コンピュータ本体側では，全方向のデータを読み取ることができます(図I-7-10)．

この図で，STROBEはCOM出力の反転されたものです．読み出し開始に当たっては，COM出力を1にしておき，以後，変化させる都度データが転送されます．

DATA VALIDは，マウスからの移動信号(パッド1，2入力レジスタに転送される)が有効な区間です．

なお，A，Bボタンの信号TRIG1とTRIG2は，STROBE信号とは無関係に入力でき，読み取りも随時可能です．

▼図I-7-9　TOWNSマウスとインタフェースのブロック図

▼図Ｉ-7-10　TOWNS マウスのタイミングチャート

T4　　　：ソフトがマウスからアドレス情報をポーリングする間隔.
TD1～4　：マウスがSTROBE信号の変化を検知してから，アドレス情報を出力するまでの時間.
TCS1～4 ：ソフトがSTROBE信号を変化させてから，マウスのアドレス情報を取り込むまでの時間.

	MIN	MAX
T1	100 (μs)	150(μs)
T2～3	50 (μs)	*1　150(μs)
T4	300(μs)	*2
TD1	—	80(μs)
TD2～4	—	40(μs)
TCS1	80(μs)　<TCS1< 100 (μs)	
TCS2～4	40(μs)<TCS2～4< 50(μs)	

*1 T2が150μs以上大きくなると，マウス側でタイムアウト処理を行い座標情報をクリアする.

*2 マウスのアドレス情報をポーリングする間隔は，マウスの移動スピードと解像度によって決定される.

7.3.2　TOWNS マウスのレジスタ

TOWNS マウスインタフェースは，TOWNS パッドインタフェースと同じレジスタを使用します.

マウスの移動量は，XY 軸方向とも 8 ビットで構成されます．マウス専用 CPU により 4 ビット(１ニブル)ずつパッド１，２入力レジスタの下位 4 ビットに転送されます．その時のデータのフォーマットは表Ｉ-7-11のようになります.

▼表Ｉ-7-11　マウスの出力データのフォーマット

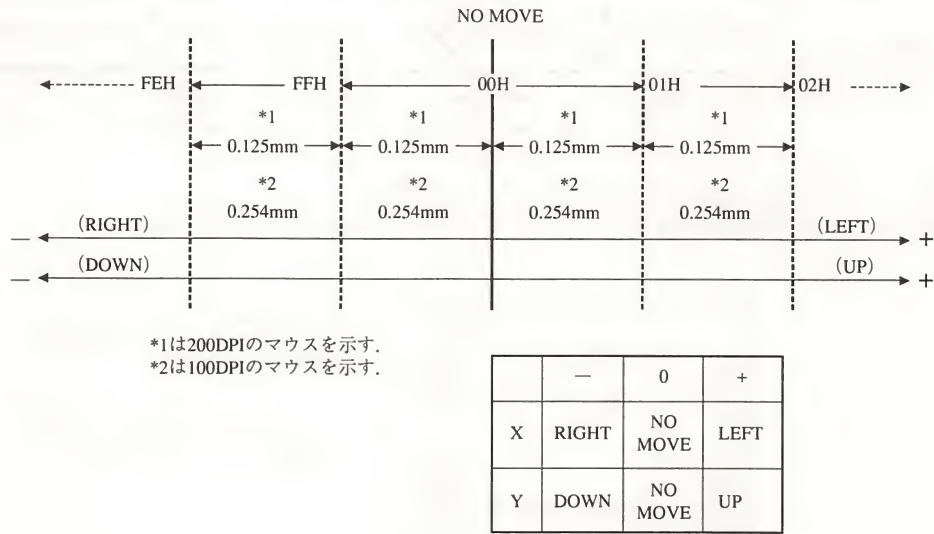
	ビット３	ビット２	ビット１	ビット０
１ニブル目	X_7	X_6	X_5	X_4
２ニブル目	X_3	X_2	X_1	X_0
３ニブル目	Y_7	Y_6	Y_5	Y_4
４ニブル目	Y_3	Y_2	Y_1	Y_0

マウスからの出力データは左の４ニブルブロックから構成される
X_7-X_0：８ビットＸ座標(２の補数表現)
１つ前のデータを出した位置からの相対座標を表し，右方向の移動が負数，左方向の移動が正数となる.
Y_7-Y_0：８ビットＹ座標
X_7-X_0と同様下方向が負数，上方向が正数となる.

また，２つのトリガボタンは，レジスタの５ビット目と６ビット目にそのフラグが転送されます．トリガボタンの状態を読み出す前に，TRIG 系出力を１にしておく必要があります.

また，マウスの移動量と出力データの関係は図 I-7-11のようになります．

パッド 1，2 出力レジスタは，規定の間隔で STROBE 信号を ON/OFF することと，トリガボタンの値を参照するのにあらかじめ TRIG1，TRIG2 に 1 をセットするために用いられます．

▼図 I-7-11　TOWNS マウスの移動量と出力データの関係

7.4　プリンタ

この節では，プリンタインタフェースの仕組みと働きについて解説します．

7.4.1　プリンタインタフェース概要

プリンタインタフェースの仕様を表 I-7-12に示します．

FMTOWNS がサポートするプリンタは FMR などと互換性があり，セントロニクス仕様に準拠しており，富士通独自の拡張がなされています．

セントロニクス仕様では，本体から 8 ビットのデータを 1 ビットにつき 1 本の信号線で並列転送します．データの流れは一方通行です．

▼表 I-7-12　プリンタインタフェースの仕様

項　　目	仕　　　様
インタフェース	セントロニクスインタフェース
データ転送	プログラム/DMA
サポート範囲	FJ拡張インタフェース

●データの流れ

本体側とプリンタ間のデータの流れを，図Ｉ-7-12に示します．

データを送出するには，まず，BUSYが０であることを確認してデータを出力し，続いてSTROBE信号を１にして，プリンタに対してデータが有効なことを宣言します．このSTROBE信号を受け取ると，プリンタ側はBUSY信号を１に保ちます．

１バイト分の受信が終了し，次のデータの受け取りが可能になった時点で，ACKNG信号を０にして本体に通知します．この時点で並行してBUSYも解除（０になる）されます．

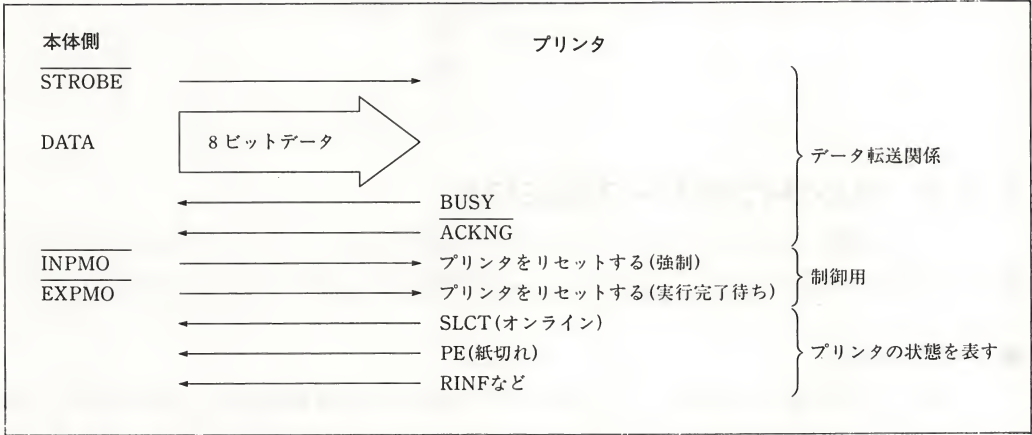
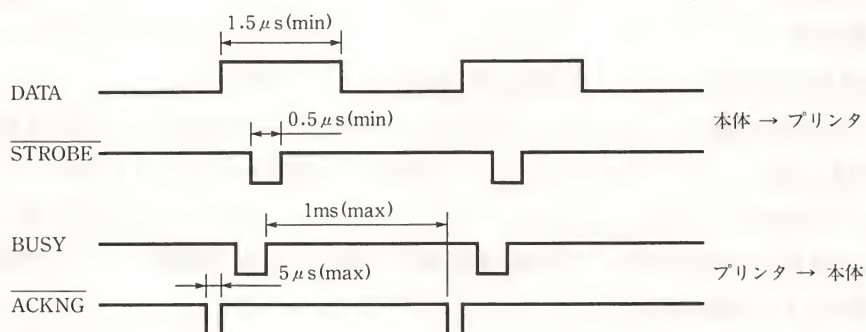
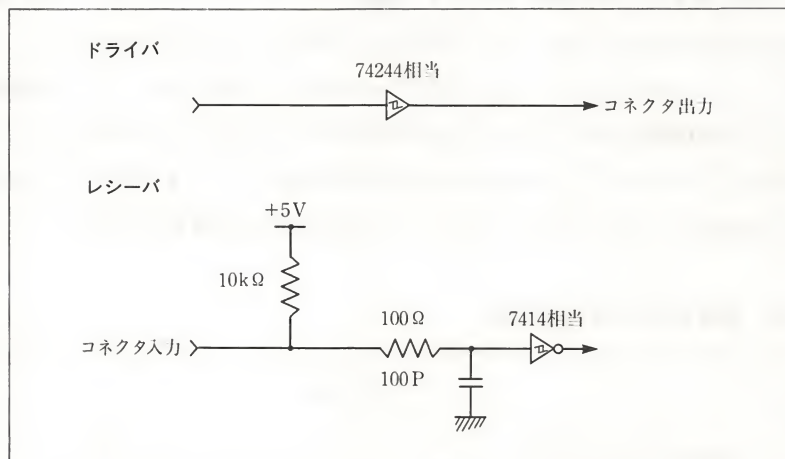
DATA，STROBE，BUSY，ACKNGのタイミングチャートを図Ｉ-7-13に示します．

８本のデータ線，STROBE線，ACKNG線，BUSY線以外には，制御用とプリンタのステータスを通知する信号線が存在します．

制御用に属するものには，プリンタをリセットするEXPM0信号（通常，電源ONで出力される強制初期化信号）やINPM0信号（プリンタを初期化する）があります．

ステータス用に属するものには，プリンタが接続（オンライン）されていることを示すSLCT信号，紙切れを通知するPE信号，エラー状態を示すRINF1～RINF3があります．なんらかのエラーが起こっている場合には，これらのステータス信号が通知され，データの送出は無視されます．また，STROBE信号に対するACKNG/BUSYの応答もなくなるので，プリンタにデータを送り出しているプログラムも永久待ち（ACKNG/BUSYを待ち続ける）となってしまいます．図Ｉ-7-14にセントロニクスインタフェースの電気的な仕様を示します．

▼図Ｉ-7-12　本体とプリンタ間の信号

本体側	方向	プリンタ	分類
$\overline{\text{STROBE}}$	→		データ転送関係
DATA	⇒（８ビットデータ）		データ転送関係
	←	BUSY	データ転送関係
	←	$\overline{\text{ACKNG}}$	データ転送関係
$\overline{\text{INPMO}}$	→	プリンタをリセットする（強制）	制御用
$\overline{\text{EXPMO}}$	→	プリンタをリセットする（実行完了待ち）	制御用
	←	SLCT（オンライン）	プリンタの状態を表す
	←	PE（紙切れ）	プリンタの状態を表す
	←	RINFなど	プリンタの状態を表す

▼図Ⅰ-7-13　各信号のタイミングチャート

▼図Ⅰ-7-14　セントロニクスインタフェースの電気的仕様

7.4.2 プリンタインタフェースのレジスタ

プリンタに関連したポートには，ステータスレジスタが2つ，データレジスタ(出力)が1つ，割り込み制御レジスタが1つ，コントロールレジスタが1つの計5つのレジスタがあります．

●データレジスタ

データレジスタ(表Ⅰ-7-13)は，プリンタに対して送るデータを書き込むレジスタです．このレジスタにデータが書き込まれると，プリンタに対してSTROBE信号が送出されます．データの取り込みが終わり，次のデータを受け取れるようになったら，本体側にACK信号を返します．

データレジスタに書き込む際に，プリンタがデータを受けとることができる状態でなければなりません．

▼表Ⅰ-7-13　データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0800H	データレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0

D7-0(bit7-0)　：プリンタに対するコマンド/データをセットする。
このレジスタにコマンド/データを書き込むと，プリンタに対するSTROBE信号が自動的に送られる。

●ステータスレジスタ1，2

ステータスレジスタ1，2(表Ⅰ-7-14，表Ⅰ-7-15)には，プリンタの状態が示されています。このレジスタを読み出せば，プリンタがデータを受け取れるかどうかが分ります。データを書き込む前には，チェックしてください。

▼表Ⅰ-7-14　ステータスレジスタ1

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0800H	ステータスレジスタ1	R	BUSY	PE	FUSE	THSN	POW	ACK	FAULT	PREADY

BUSY(bit7)　：プリンタが動作中であることを示す。
0＝データ待ち
1＝動作中

PE(bit6)　：プリンタ用紙が残り少なくなったことを示す。
0＝用紙あり
1＝用紙なし

FUSE(bit5)　：＊プリンタのヒューズ断を示す。
0＝正常
1＝ヒューズ切れ

THSN(bit4)　：＊プリンタの印字ヘッドの異常温度検出を示す。
0＝正常
1＝温度異常

POW(bit3)　：＊プリンタの電源ON/OFF状態を示す。
0＝OFF
1＝ON

ACK(bit2)　：コマンド/データに対する応答があったことを示す。
ステータスレジスタ2のリードでリセットする。
0＝応答なし
1＝応答あり

FAULT(bit1)　：プリンタのアラーム，または，オフライン状態を示す。
ステータスレジスタ2のリードでリセットする。
0＝正常
1＝アラーム，又は，オフライン

PREADY(bit0)　：プリンタのデータ受信状態を示す。DMA，および，割り込みを用いないでデータ転送する場合，このビットが1であることを確認すること。
0＝プリンタへデータ転送不可
1＝プリンタへデータ転送可能

説明の最初に＊の付くものは，プリンタによりサポートされない場合がある(詳細は各プリンタの仕様書を参照)。この場合，そのビットに対して読み出した値は不定。

▼表I-7-15　ステータスレジスタ2

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0802H	ステータスレジスタ2	R	不　定				SLCT	RINF3	RINF2	RINF1

SLCT(bit3)　：　プリンタのセレクト状態を示す.
　0＝オフライン状態
　1＝オンライン状態

RINF3-1(bit2-0)　：＊プリンタのエラー/アラーム状態を示す.おのおの,その状態に伴い,FAULT,BUSYビットがともに出力される.
（詳細は,各プリンタの仕様書を参照のこと）

このレジスタは,ステータスレジスタ1のFAULT＝1のとき以外読まないこと.
説明の最初に＊の付くものは,プリンタによりサポートされない場合がある(詳細は各プリンタの仕様書を参照).この場合,そのビットに対して読み出した値は不定.

●割り込み制御レジスタ

割り込み制御レジスタ(表I-7-16)では,ACKまたはFAULT信号によって,CPUに対して割り込みをかけるように設定することができます.

一般にプリンタのデータは,行単位に出力されることが多く,最後の行が出力された後はしばらく使われないことが少なくありません.そのようなときプリンタをオフラインにしたり,用紙をはずしたりしただけで割り込みが発生すると不都合なので,プログラムでは遊休時に割り込みを禁止するようにしておかなければなりません.

▼表I-7-16　割り込み制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0804H	割り込み制御レジスタ	W	0	0	0	0	0	0	ACK MSK	FLT MSK

ACKMSK(bit1)　：ACK信号による割り込みを制御する.
　0＝割り込み禁止
　1＝割り込み許可

FLTMSK(bit0)　：FAULT信号による割り込みを制御する.
　0＝割り込み禁止
　1＝割り込み許可

●コントロールレジスタ

コントロールレジスタ(表I-7-17)では,プリンタの初期化とDMAの起動を指示します.EXPRMとINPRMは,プリンタに対して初期化を指示するので,通常は0にしておいてください.

▼表 I-7-17　コントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0802H	コントロールレジスタ	W	0	0	0	0	0	EXPRM	INPRM	DMA

EXPRM(bit2)　：＊プリンタへのイニシャライズを指示する．
このビットはセントロニクスインタフェース上のエクスプライム信号であり，このビットに1を書くとプリンタはそのとき実行中の動作を中断する．一定時間(プリンタにより異なるので各々の仕様書を参照のこと)ののちに0を書いたときに，イニシャル動作を実行する．
定常状態では，0にしておくこと．

INPRM(bit1)　：プリンタへのイニシャライズを指示する．
このビットはセントロニクスインタフェース上のインプライム信号であり，このビットに1を書くとプリンタはそのとき実行中の動作を正常に終了する．一定時間(プリンタにより異なるので各々の仕様書を参照のこと)ののちに0を書いたときに，イニシャル動作を実行する．
定常状態では，0にしておくこと．

DMA(bit0)　：プリンタ用のDMAをスタートさせる．
DMA終了後，このビットは自動的に0になる．
1＝DMA開始

DMA開始後，終了するまではこのレジスタへの書き込みを行わないこと．
説明の最初に＊の付くものは，プリンタによりサポートされない場合がある(詳細は各プリンタの仕様書を参照)．この場合，そのビットに書き込む場合は0にすること．

7.5 フロッピィディスクドライブ

FMTOWNSには，3.5インチフロッピィディスクドライブが内蔵されています．FDC(フロッピィディスクコントローラ)は，MB8877Aを採用しており，ディスクドライブの制御に使われています．この節では，FDCを使った，ハードウェアレベルのディスクの読み書きの仕組みについて解説します．

7.5.1 ディスクドライブの仕様

内蔵のフロッピィディスクドライブの仕様とフロッピィディスクコントローラの仕様を，表I-7-18，表I-7-19に示します．

ディスクドライブは，2HD/2DD両用タイプであり，どちらのメディアでも読み出し書き込みが可能です．

また，2Dのメディアは書き込みはできませんが，読み出しは可能です．

FMTOWNSに内蔵されているディスクは3.5インチですが，増設の5インチの2台は，同じコントローラ(FDインタフェース)を使って制御できます．

▼表 I-7-18 内蔵フロッピィディスクドライブの仕様

項	項目	仕様	
		2DDモード	2HDモード
1	容量	記憶容量/ドライブ Unformat：1.0MB Format：655KB 記憶容量/トラック Unformat：6,250B Format：4,096B	記憶容量/ドライブ Unformat：1.6MB Format：1.025KB 記憶容量/トラック Unformat：10,416B Format：6,656B
2	回転速度	300rpm	360rpm
3	記憶密度	8,717bpi	14,184bpi
4	シリンダ数	80	77
5	転送速度	31.25KB/S	62.5KB/S
6	ヘッド数	2	2
7	トラック密度	135TPI	135TPI
8	モータ起動時間	1,000mS以下	1,000mS以上
9	シーク速度	ポジショニングタイム MIN 18mS以下 TYP 95mS以下 MAX 252mS以下	ポジショニングタイム MIN 18mS以下 TYP 91mS以下 MAX 243mS以下
10	電源仕様	＋５Ｖ±５％ MAX 1.2A	＋５Ｖ±５％ MAX 1.2A

▼表 I-7-19 フロッピィディスクコントローラの仕様

項目	仕様
コントローラ	MB8877A
サポート範囲	3.5インチ 2HD/2DD 3.5インチ 2D（リードのみ） 5¼インチ 2HD/2DD 5¼インチ 2D（リードのみ）
ドライブ数	最大４台
レディチェック	選択されている１台のみチェック可能
データ転送	DMA
その他	モータON/OFF可能

サポート範囲は，内蔵ドライブの外に，外付けドライブも含めて対応できる記録形式を示す.

7.5.2 フロッピィディスクのフォーマット

ディスクを同心円上に分割したものをトラック．さらに，等角度で分割したものをセクタといいます(図I-7-15)．データをディスクに記録する際には，セクタが最小単位になります．

セクタの中は，セクタの読み書き時に同期をとるための領域，セクタのIDを格納する領域，

データが格納されている領域などに分れています．

トラックとセクタのデータ構造を，図Ｉ-7-16に示します．この図中，DATA フィールドにある，DATA の部分に正味のデータが格納されます．

FMTOWNS の基本OS である TownsOS では，MS-DOS と同じフォーマットを採用し，FMR などの他機種とデータの互換性がとれるようにしています．

TownsOS のフォーマットを，表Ｉ-7-20に示します．

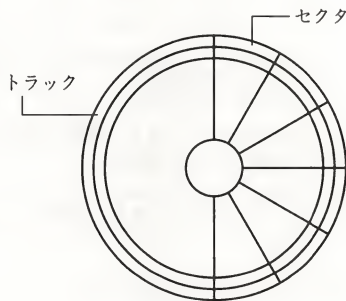
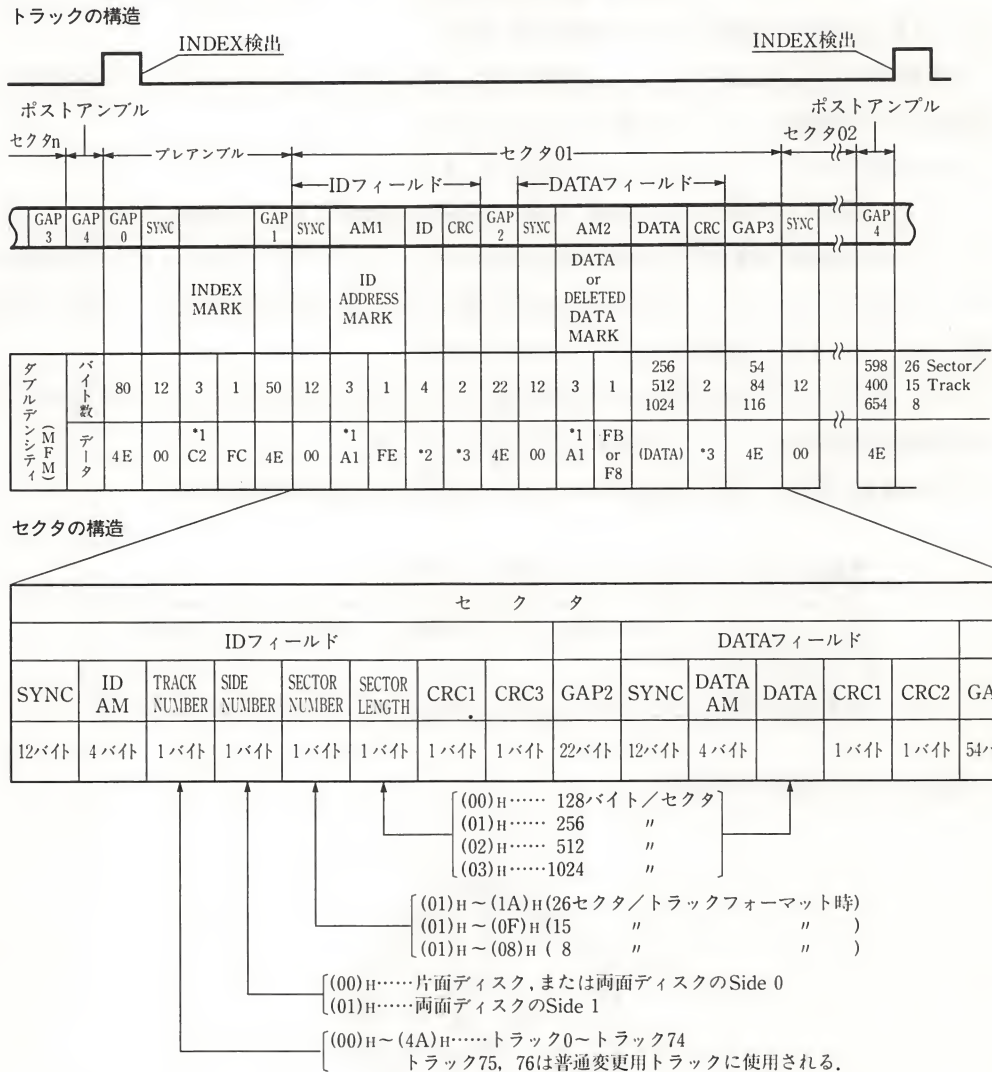
ディスクのトラック数，セクタの数，セクタの長さは，使用する OS が採用しているフォーマットによって異なりますが，FMTOWNS の FDC(フロッピィディスクコントローラ)を使って，ハードウェアレベルでデータの読み書きを行う場合には，TownsOS だけでなく，FM77 やその他のフォーマットにも対応できるようになっています．

シリンダとは，ディスクを同心円状に何分割しているかを示すもので，トラック数はディスク両面が使用可能なため，シリンダ数×２となっています．

セクタ長は，図Ｉ-7-16の DATA フィールドの DATA の部分に相当します．

▼表Ｉ-7-20　フロッピィディスクのフォーマット

メディア	シリンダ数	トラック数	セクタ数	セクタ長
2HD(1MB)	77	154	8	1024
2DD(640KB)	80	160	8	512

▼図Ｉ-7-15　セクタとトラック

▼図Ⅰ-7-16　トラックとセクタのデータ構造

*1………ミッシングクロックを持っていることを示す．
*2………IDフィールドを示す．
*3………Cyclic Redundancy Check　多項式$G(X)=X^0+X^5+X^{12}+X^{16}$

7.5.3　フロッピィディスクドライブの基本動作

ここでは，フロッピィディスクドライブが，どのような動作を行っているかについて説明します．以下に説明する動作は，FDCの助けを借りて行います．

フロッピィディスクドライブの動作は，基本的にディスクの読み書きと，ヘッドの移動だけです．

●ディスクの読み書き

ディスクの読み書きには次の５つの形式があります．

ライトトラック

１トラックに対して書き込みを行うことです．フォーマット時などには，これを使って，各セクタのIDフィールドに，サイド番号(面番号)，トラック番号，セクタ番号を書き込み，トラック内にセクタを割り付けます．

リードトラック

１トラックの内容を一度に読み出すことです．ディスクの診断時などに使われます．

リードトラックや，ライトトラックの際は，トラックの始まりをインデックス孔検出信号によって識別します．

リードデータとライトデータ

セクタの正味のデータ(セクタ長に相当する部分)ごとに，読み書きすることです．

リードデータ，ライトデータの実行は，IDフィールドを読みながら目的のセクタを捜し，見つかった時点で，そのデータフィールドの位置から読み出し，書き込みを行います．

リードアドレス

IDフィールドの内容を読み出すことです．

●ヘッドの移動

ヘッドの動作には，リストア，ステップ，シークの３種類があります．

図Ｉ-7-17に，ヘッドの移動動作と位置関係を表します．

リストア

現在のヘッドのあるトラック位置から，最外周(トラック０)へ移動する動作をいいます．

ステップ

現在のヘッドのあるトラック位置から，隣接トラックへ移動する動作をステップといいます．外周，内周どちらにも移動ができます．外周へのステップをステップアウト，内周へのステップをステップインといいます．

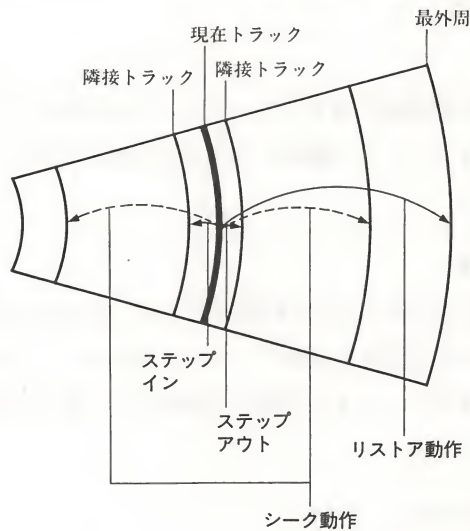
シーク

任意のトラック番号の位置にヘッドを移動させることをシークといいます.

シーク時には，現在のトラック位置と行き先のトラック番号の差が正ならば内周方向へ，負ならば外周方向へ移動します(最内周が最大トラック番号，最外周トラックは0). 差が0であるときにはシーク動作は行われません.

シーク動作は，ステップ動作の繰り返しとして行われます. 例えば，トラック間の差が2であれば，2回のステップ動作が行われることになります.

システム起動時には，トラックレジスタ(現在のトラック中アドレスを示す)などの値が設定されていないので，まず，リストア動作を行います. その後，ステップ動作，シーク動作が可能となります.

▼図I-7-17　ヘッドの移動動作と位置関係

7.5.4 FDCのレジスタ

FDCを使って，フロッピィディスクドライブを制御することができます.

FDCには，表I-7-21に示すようなレジスタがあります.

また，この外にフロッピィディスクドライブの制御に関わっているフロッピィディスクドライブ制御用補助レジスタ(表I-7-22)があります.

▼表Ⅰ-7-21　FDC(MB8877A)のレジスタ

I/Oアドレス	R/W	レジスタ名
0200H	R	ステータスレジスタ
	W	コマンドレジスタ
0202H	R/W	トラックレジスタ
0204H	R/W	セクタレジスタ
0205H	R/W	データレジスタ

▼表Ⅰ-7-22　フロッピィディスクドライブ制御用補助レジスタ一覧

I/Oアドレス	R/W	レジスタ名
0208H	R	ドライブステータスレジスタ
	W	ドライブコントロールレジスタ
020CH	W	ドライブセレクトレジスタ
020EH	R/W	ドライブスイッチレジスタ

●コマンドレジスタ

コマンドレジスタ(表Ⅰ-7-23)は，フロッピィディスク制御コマンドを書き込んで与えるためのものです．

コマンドにはⅠからⅣまでの４タイプがあり，送出したコマンドはタイプごとにステータスレジスタにその結果が返ってきます．

与えるパラメータは，タイプⅠのコマンドでは，u/h/V/r1・r0，タイプⅡではm/S/E/C/a0，タイプⅢではEのみ，タイプⅣではI3～I0となっています．

タイプⅣのコマンドを送出し，インタラプトフラグであるI3～I0のいずれかに１を指定した場合，強制割り込みが生じます．I0はREADY入力の立上り時，I1はREADY入力の立ち下がり時，I2は各インデックスパルス，I3は無条件に割り込みが起こります．

各コマンドコードのフラグについて説明します．

h(ヘッドロード)とは，ヘッドをディスクの表面に接触させることです．V(照合)とは，IDフィールドのチェックを行うかどうかです．u(トラックレジスタの更新)はトラックレジスタの値を更新するかどうかを決めるものです．

r0/r1(ステップレート)は，ステップパルスの間隔を示します．

FMTOWNSでは，FDCクロックの値により，1MHzでは１，2MHzでは０を指定します．FDCクロックについては後述のドライブコントロールレジスタ(表Ⅰ-7-26)のCLKSELの説明を参照してください．

m(マルチレコード)は，連続して複数レコードを処理することです．

a0(アドレスマーク)は，不良セクタのとき１を書き，その他は０を書きます．

S(サイド番号)は，ディスクの面番号(０，１)を示します．

E(15msディレイイネーブル)は，ヘッドロード時間に15msを自動加算する機能です．

C(サイド比較)は，サイドの値が正しいかどうかのチェックを行うものです．

▼表I-7-23 コマンドレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0200H	コマンドレジスタ	W	D7	D6	D5	D4	D3	D2	D1	D0

フロッピィディスクドライブにコマンドを与える．次のようなタイプがある．

タイプ	名　称	コ　ー　ド	動　作　概　要
I	リストア	0 0 0 0 h V r_1 r_0	ヘッドをトラック0へ移動する
	シーク	0 0 0 1 h V r_1 r_0	目的のトラックへ，ヘッドを移動する
	ステップ	0 0 1 u h V r_1 r_0	ヘッドを1トラック移動する
	ステップイン	0 1 0 u h V r_1 r_0	ヘッドを1トラック内側へ移動する
	ステップアウト	0 1 1 u h V r_1 r_0	ヘッドを1トラック外側へ移動する
II	リードデータ	1 0 0 m S E C 0	ディスクのデータ(データフィールド)を読む
	ライトデータ	1 0 1 m S E C a_0	ディスク(データフィールド)へデータを書く
III	リードアドレス	1 1 0 0 0 E 0 0	ディスクの1Dフィールドを読む
	リードトラック	1 1 1 0 0 E 0 0	ディスクの1トラックの全データを読む
	ライトトラック	1 1 1 1 0 E 0 0	ディスクの1トラックの全データを書く
IV	フォースインタラプト	1 1 0 1 I_3 I_2 I_1 I_0	割り込み(IRQ)を発生させる

フラグ名称

r_1, r_0　：ステップレート(Step Rate)
V　：照合(Verify)
h　：ヘッドロード(Head Load)
u　：トラックレジスタの更新
m　：マルチレコード(Multi-Record)

a_0　：アドレスマーク(ADDRESS MARK)
S　：サイド番号(Side-Number)
E　：15msディレイイネーブル
(15ms Delay Enable)
C　：サイド比較(Side Compare)
I_3〜I_0　：割り込み(Interrupt)

●ステータスレジスタ

ステータスレジスタ(表I-7-24)は，コマンドレジスタに書き込んだコマンドに対して，ディスクドライブの状態を返すレジスタです．コマンドのタイプによって，返される値の意味が異なります．

ビット7は，すべてのコマンドに対して共通で，ドライブが動作可能かどうか(1ならば不可能)を意味します．

ビット6はディスク書き込みタイプのコマンドのみ意味があり，このフラグが1であるときは，ライトプロテクトがかかっていることを示します．

ビット5は場合によって意味が違いますが，動作に失敗したときに1となります．HEAD ENGAGEDが1のときにはヘッドロードされていることを示します．RECORD TYPEは，不良セクタの場合に1となります．

ビット4が1のときはヘッド移動時にシークエラー(あるいはセクタが見つからない)が生じたことを示します．

ビット3が1のときは，ディスクに物理的エラーがある(CRCエラー)ことを示します．

ビット２が１のときは，タイプＩの場合は０トラックに移動したことを示し，他のタイプのコマンドの場合はデータの転送が間に合わなくて失われたことを示します．

ビット１が１のときは，タイプＩコマンドの場合インデックスホールの検出を意味します．他のタイプの場合には次のデータを要求することを意味します．

ビット０はどのタイプでも共通で，このフラグが１のときはFDCがコマンド実行中であり，次のコマンドが受け付けられないことを示します．

表中のマスタリセットとは，FDCに対するハードウェアのリセット信号のことです．リセットがかかると，それまでの動作の解除と同時に自動的にリストアが実行されます．

▼表Ｉ-7-24　ステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0200H	ステータスレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0

コマンドに対してドライブの状態を返す．各ビットは次のような意味がある．

タイプ	コマンド	7	6	5	4	3	2	1	0
I	Ｉのタイプのすべてのコマンド	NOT READY	WRITE PROTECT	HEAD ENGAGED	SEEK ERROR	CRC ERROR	TRACK00	INDEX	BUSY
II	リードデータ	NOT READY	0	RECORD TYPE	RECORD NOT FOUND	CRC ERROR	LOST DATA	DATA REQUEST	BUSY
	ライトデータ	NOT READY	WRITE PROTECT	WRITE FAULT	RECORD NOT FOUND	CRC ERROR	LOST DATA	DATA REQUEST	BUSY
III	リードアドレス	NOT READY	0	0	RECORD NOT FOUND	CRC ERROR	LOST DATA	DATA REQUEST	BUSY
	リードトラック	NOT READY	0	0	0	0	LOST DATA	DATA REQUEST	BUSY
	ライトトラック	NOT READY	WRITE PROTECT	WRITE FAULT	0	0	LOST DATA	DATA REQUEST	BUSY
IV	他のコマンド実行中の場合	（今まで実行していたコマンドのステータスビットと同様の意味）							0
	実行中のコマンドがない場合	NOT READY	WRITE PROTECT	HEAD ENGAGED	0	0	TRACK00	INDEX	0
マスタリセット		（タイプＩコマンドに準ずる）							

●トラックレジスタ

トラックレジスタ(表I-7-25)は，読み出し／書き込み両用レジスタであり，現在のヘッド位置が8ビットで格納されています．0～79までの数値を2進値で示します．この値はリストアで0になり，ステップインで1加算，ステップアウトで1減算されます．このようなトラックレジスタの自動更新を禁止することもできます．その場合，プログラムでトラック値を書き込まなければなりません．

▼表I-7-25　トラックレジスタ，セクタレジスタ，データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0202H	トラックレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0204H	セクタレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0
0206H	データレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0

●セクタレジスタ

セクタレジスタ(表I-7-25)は，読み出し／書き込み両用レジスタで，リードデータ，ライトデータでは対象のセクタ位置を指定します．セクタ値は1～16までの数値を2進値で示します．

リードアドレス時には，IDフィールドのトラック番号が入ります．

●データレジスタ

データレジスタ(表I-7-25)は，読み出し／書き込み両用レジスタです．このレジスタには，ディスクから読み出したデータ(読み出し時)，あるいはディスクに書き込むデータ(書き込み時)が格納されます．

シーク時には，事前にこのレジスタに目的トラック値を書き込みます．そして，シークコマンドを実行すると，トラックレジスタの値と目的トラック値を比較しながら，ステップインまたは，ステップアウトを繰り返し，トラックレジスタの値と等しくなるまで，ヘッドを移動します．

●ドライブステータスレジスタ

ドライブステータスレジスタ(表I-7-26)は，ドライブの状態を参照するもので，拡張ドライブの種別，選択されたドライブがレディ(動作可能状態)であるかどうかを知ることができます．

このレジスタのビット3～7は不定，ビット0は常に1となっています．

▼表 I-7-26　ドライブステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0208H	ドライブステータスレジスタ	R	不　定					3.5 FDD	FREADY	1

3.5 FDD (bit2)　：拡張ドライブの種別を示す．
0＝5.25インチ
1＝3.5 インチ

FREADY (bit1)　：選択されたドライブがレディであることを示す．
0＝ノットレディ
1＝レディ

●ドライブコントロールレジスタ

ドライブコントロールレジスタ(表 I-7-27)は，FDC のクロック，モータの ON/OFF，ディスクのサイド，記録密度，割り込みマスクの指定をします．

▼表 I-7-27　ドライブコントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0208H	ドライブコントロールレジスタ	W	0	0	CLK SEL	MOTOR	0	HD1 SEL	DDEN	IRQ MSK

CLKSEL (bit5)　：FDCに与えるクロックを指定する．
0＝2 MHz(3.5インチ，5インチ2HD)
1＝1 MHz(3.5インチ，5インチ2D/2DD)
シーク動作時はこのビットを0としておくことにより，高速シークが可能である．
リード/ライト時は制御対象となるメディアに応じた設定(上記のとおり)としなければならない．

MOTOR (bit4)　：3.5インチ，5インチドライブのモータを制御する．
0＝OFF(停止)
1＝ON(回転)

HDISEL (bit2)　：リード/ライトの対象となるメディアの面を指定する．
0＝サイド1
1＝サイド0

DDEN (bit1)　：メディアの記録方式を指定する．
0＝単密度
1＝倍密度(2D/2DD/2HD)

IRQMSK (bit0)　：FDCからの割り込みを制御する．
0＝割り込み禁止
1＝割り込み許可

●ドライブセレクトレジスタ

ドライブセレクトレジスタ(表 I-7-28)は，ドライブコントロールレジスタの機能補完と，ドライブの選択を行います．モータの回転数，インユース信号の制御，DSL0〜DSL3 のドライブセレクト情報を出力するようになっています．

インユースの制御は，入出力の直前に1(点灯)にし，終了後に0(消灯)にもどすようにします．

▼表I-7-28　ドライブセレクトレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
020CH	ドライブセレクトレジスタ	W	0	HISPD	0	INUSE	DSL3	DSL2	DSL1	DSL0

HISPD(bit6)　：3.5インチ，5インチドライブの回転速度を指定する．
0＝300rpm(2D/2DD)
1＝360rpm(2HD)

INUSE(bit4)　：ドライブのドアロック機構，および，ランプの制御を行う．
3.5インチ，5インチドライブの場合
0＝DSLをオフにしたときラッチ解除を行い，表示ランプを消灯し，DSL信号をオンにしても，表示ランプを点灯しない．
1＝DSLをオンにしたときこれをラッチし，ドライブ使用中を示す表示ランプを点灯し，DSL信号オフ後もこの状態を保ち点灯し続ける．

DSL3-0(bit3-0)　：本体内蔵，及び，増設フロッピィのドライブを指定する．
1＝各ビットに応じたドライブが選択される．
同時に2つ以上のビットを1にしてはならない．

HISPD(bit6)，INUSE(bit4)は，DSL3-0(bit3-0)を1にしたときラッチされる．
このため，HISPDまたはINUSEをセット/リセットしてからドライブ指定を行う(レジスタへの書き込みを二度行う)こと．

●ドライブスイッチレジスタ

ドライブスイッチレジスタ(表I-7-29)の最下位ビットのみに意味があり，内蔵ドライブと外付けドライブのドライブ番号を交換することができます．

▼表I-7-29　ドライブスイッチレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
020EH	ドライブスイッチレジスタ	R/W	0	0	0	0	0	0	0	DRV CHG

DRVCHG(bit0)　：FDのドライブ番号について内蔵FDと外付けFDを入れ換える．
0＝DSL 0がドライブ0に対応する(内蔵3.5インチFD)．
DSL 1がドライブ1に対応する(内蔵3.5インチFD)．
DSL 2がドライブ2に対応する(外付け5インチFD/3.5インチFD)．
DSL 3がドライブ3に対応する(外付け5インチFD/3.5インチFD)．
1＝DSL 0がドライブ2に対応する(外付け5インチFD/3.5インチFD)．
DSL 1がドライブ3に対応する(外付け5インチFD/3.5インチFD)．
DSL 2がドライブ0に対応する(内蔵3.5インチFD)．
DSL 3がドライブ1に対応する(内蔵3.5インチFD)．

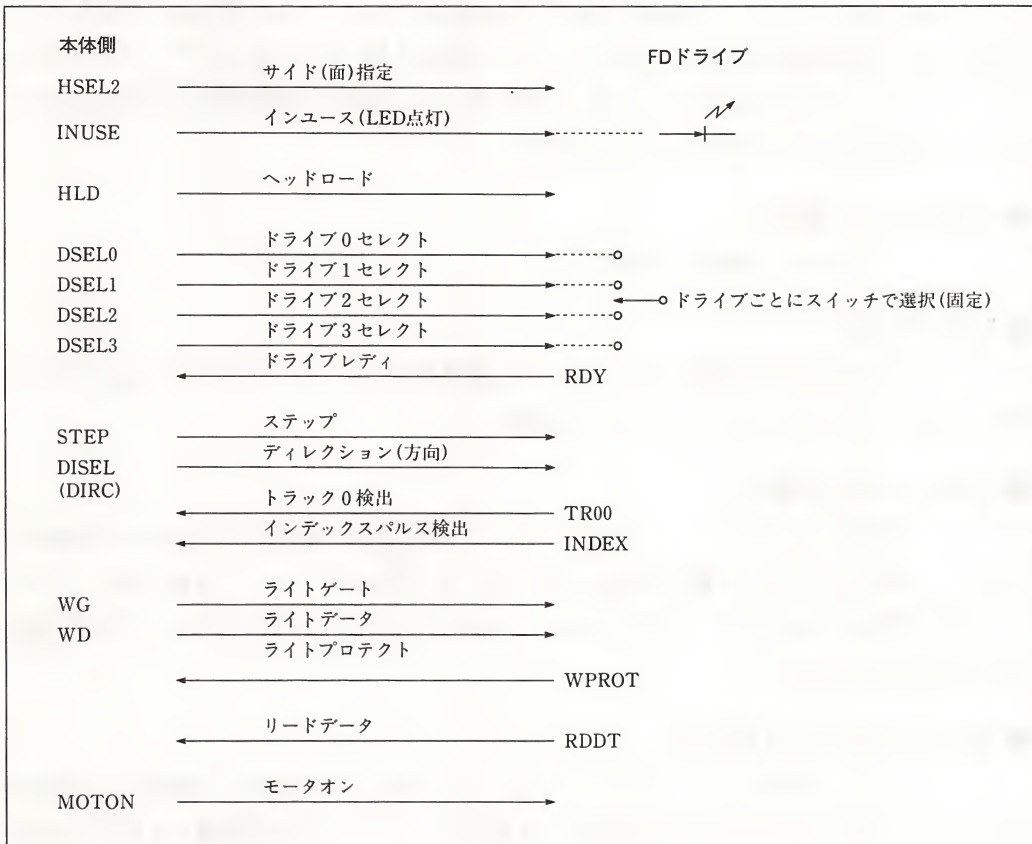
7.5.5 フロッピィディスクドライブ制御の信号線

フロッピィディスクドライブと本体(FDC)は図I-7-18のような信号線によって結ばれています．

この信号線を経由したデータのやり取りは，FDCが行っているので，ユーザーは意識する必要はありません．以下の説明は，拡張ドライブ接続時などの参考にしてください．

一般に，レジスタなどの設定値に対してビット値が反転しているので，注意が必要です．

▼図I-7-18　フロッピィディスクドライブの信号線

本体側	方向	信号	FDドライブ
HSEL2	→	サイド(面)指定	
INUSE	→	インユース(LED点灯)	
HLD	→	ヘッドロード	
DSEL0	→	ドライブ0セレクト	
DSEL1	→	ドライブ1セレクト	ドライブごとにスイッチで選択(固定)
DSEL2	→	ドライブ2セレクト	
DSEL3	→	ドライブ3セレクト	
	←	ドライブレディ	RDY
STEP	→	ステップ	
DISEL (DIRC)	→	ディレクション(方向)	
	←	トラック0検出	TR00
	←	インデックスパルス検出	INDEX
WG	→	ライトゲート	
WD	→	ライトデータ	
	←	ライトプロテクト	WPROT
	←	リードデータ	RDDT
MOTON	→	モータオン	

● HSEL2(サイド)

フロッピィディスクのどちらの面をアクセスするかを指定します．1の場合はサイド0，0の場合はサイド1に対応します．

● INUSE(インユース)

使用中のドライブのインジケータを点灯させるための信号です．

ドライブに対するアクセスが行われる前に出力を0にして点灯させ，終了後に出力を1にして消灯させます．この操作はソフトウェアによって対応しなければなりません．

ドライブセレクトレジスタの同名のビットと反対の値が出力されます.

● HLD(ヘッドロード)

ヘッドを接触させるための信号線です. FMTOWNSの3.5インチフロッピィディスクではディスク挿入と同時にヘッドロードが行われます. したがって, ドライブがHLDの値にかかわらずヘッドロードします. 5インチのディスクの場合には, 書き込みや読み取りの際のみ, ヘッドロードさせます.

● DSEL0(ドライブセレクト0)～DSEL3(ドライブセレクト3)

0～3番目のどのドライブを駆動するかを決める出力です. ドライブ側では内蔵のスイッチで指定された番号に対応します. この信号に対する応答がRDY(ドライブレディ)です. ドライブセレクトは, 4本のうち同時には1本しか働きません. 対応する番号の線に0が出力されたとき, その番号のドライブがセレクトされます.

● STEP(ステップ動作)

1トラックずつヘッドを動かす信号です.

● DISEL(DIRC)

ステップの方向を示す信号線です. 1のときには外周方向へ(ステップアウト), 0のときは内周方向に(ステップイン), 1トラック分移動します.

● TR00(トラック0信号)

リストア動作を行ったりして, ヘッドがトラック0に位置した場合, トラック0信号が出力されます. 例えば, リストア動作の場合には, FDCは外周に向けてシークを繰り返し, トラック0信号を検出するまでシークを行います. この信号は0でトラック0を表し, その他の場合は1が出力されます.

● INDEX(インデックスパルス)

フロッピィディスクのインデックスホールをドライブのセンサが検出した場合に0になる信号です. インデックスパルス信号の検出は, すなわち, トラックの先頭位置にきたことを意味します.

● WG(ライトゲート)

データの書き込みのタイミングを与えます. この信号線が0のとき, フロッピィディスクへの書き込みが許可されます. ディスクにライトプロテクトが施されている場合には, WPROT(ライトプロテクト信号)がFDCに通知され, ディスクへの書き込みが禁止されます.

● WD(ライトデータ)

フロッピィディスクへの書き込みデータを与えます.

● RDDT（リードデータ）

フロッピィディスクから読み出したデータを FDC に転送します．FDC は RDDT 線から受け取ったビット信号をバイト単位に組み直し，CPU に転送します．

● MOTON（モータオン）

ドライブのモータを制御します．この出力が０のとき，モータが回転し，１のとき停止します．フロッピィディスクドライブをしばらく使わないときはモータを止めておくことにより，ドライブの寿命を延ばすことも可能ですが，一度，止まったモータは再起動に時間がかかり，データアクセスが遅くなります．

プログラムでは，ファイルオープン時に ON にし，クローズで OFF にするのがよいでしょう．

7.5.6　増設ドライブについて

内蔵フロッピィディスクドライブの外に，外付けドライブを拡張する場合には，本体裏面にあるフロッピィコネクタを通じて接続します．メーカー指定の拡張ドライブを接続する場合には，単にケーブルで接続すれば問題ありません．

7.6　ハードディスク

FM TOWNS では，ハードディスクは SCSI インタフェースで接続されます．

この節では，SCSI インタフェースの概要とハードディスクのレジスタについて解説します．

7.6.1　ハードディスクの仕様

ハードディスクの仕様を表 I－7－30に示します．

FM TOWNS に接続できるハードディスクは，SCSI 仕様に準拠したインタフェースで本体と接続されます．SCSI とは，Small Computer System Interface の略であり，独立したシステム間のインタフェースを規定した規格の１つです．データ転送は非同期で行われており，DMA によるデータ転送も可能です．

▼表 I-7-30　ハードディスクの仕様

項　目	仕　　様
インタフェース	外付　SCSI
容　　量	外付　２０/４０/６７/１３０MB
サ　イ　ズ	外付　３.５/５インチ

7.6.2 SCSIとは

SCSIは，もともと磁気ディスクとホストコンピュータ間のデータ転送用に作られた規格ですが，磁気ディスクだけにとどまらず，各種の装置間でデータのやり取りをすることができる仕様となっています．

SCSIでは，8ビットのデータバスを使用しており，8個の装置を接続することができます．各装置のCPU間で通信をしながらデータのやり取りを行います．したがって，接続する装置は，インテリジェントであることが前提になります．

SCSIは，多量のデータを高速に転送できます．また，複数のホストと複数のデバイスを接続できるので，将来的にみても拡張性が高いインタフェースであるといえます．

8ビットデータバスでは，単なるデータの外にコマンドや，その応答であるステータス，制御データとしてのメッセージなどが転送されます．

7.6.3 ハードディスクのレジスタ

SCSIインタフェースを介したハードディスクの制御は，コントロールレジスタ(表I-7-31)，動作状態を参照するステータスレジスタ(表I-7-32)，データの受渡しを行うデータレジスタ(表I-7-33)の3つのレジスタへの書き込みと読み出しによって行います．

▼表I-7-31　コントロールレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C32H	コントロールレジスタ	W	WEN	IMSK	0	ATN	0	SEL	DMAE	RST

WEN(bit7)　：SCSIバスへのデータ，コントロール信号の出力を制御する．
　0＝出力を禁止する
　1＝出力を許可する

IMSK(bit6)　：ステータスレジスタのINT(bit1)の割り込みを制御する．
　0＝割り込み許可
　1＝割り込み禁止

ATN(bit4)　：周辺装置に対して何らかのメッセージがあることを示す．
　0＝メッセージなし
　1＝メッセージあり

SEL(bit2)　：SCSIバスのSEL信号の制御を行う．
　0＝OFF
　1＝ON

DMAE(bit1)　：DMA転送を制御する．
　0＝DMA転送を禁止する
　1＝DMA転送を行う

RST(bit0)　：SCSIコネクタに接続しているすべての周辺装置をリセットする．
　0＝リセット解除
　1＝リセット(25μs以上後に0にもどすこと)

▼表 I-7-32　ステータスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C32H	ステータスレジスタ	R	REQ	I/O	MSG	C/D	BUSY	不定	INT	PERR

REQ (bit7)　：Information Transferフェーズにおける転送要求を示す．
0＝転送要求なし
1＝転送要求あり

I/O (bit6)　：データの入出力の方向を示す．
0＝出力
1＝入力

MSG (bit5)　：データレジスタの内容がメッセージであるか，データであるかを示す．
0＝データ
1＝メッセージ

C/D (bit4)　：データレジスタの内容がコントロール情報であるか，データであるかを示す．
0＝データ
1＝コントロール情報(コマンド，ステータス，メッセージ)

BUSY (bit3)　：SCSIバスの状態を示す．
0＝解放されている
1＝使用中である

INT (bit1)　：割り込みが発生したことを示す．
0＝割り込みなし
1＝割り込みあり
この割り込みは，Command, Status, Messageのいずれかのフェーズに移行し，REQ信号が１になったときに発生する．ただし，コントロールレジスタのIMSK (bit6)に０を書くことによりマスクすることができる．

PERR (bit0)　：周辺装置からのデータのパリティエラーを示す．
0＝パリティエラーなし
1＝パリティエラーあり
ステータスレジスタを読むと，このビットは０になる．

C/D	MSG	I/O	データレジスタの内容	入出力	フェーズ
0	0	0	データ	OUT	Data Outフェーズ ┐Data
0	0	1	データ	IN	Data In フェーズ ┘フェーズ
0	1	0		OUT	予約済
0	1	1		IN	予約済
1	0	0	コマンド	OUT	Commandフェーズ
1	0	1	ステータス	IN	Status フェーズ
1	1	0	メッセージ	OUT	Message Outフェーズ ┐Message
1	1	1	メッセージ	IN	Message In フェーズ ┘フェーズ

▼表 I-7-33　データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C30H	データレジスタ	R/W	D7	D6	D5	D4	D3	D2	D1	D0

D7-0 (bit7-0)　：周辺装置とのデータ，コマンド，ステータスの受け渡しを行う．
このレジスタのリード/ライトによってACKが送出されるので，不必要にリード/ライトを行わないこと．

なお，表I-7-32中のフェーズは，バスが，今何をしているか(使用状態)を表すものです．ある時点を捉えてみると，バスは1つのフェーズで動作していることになります．

ここでは，コンピュータとデバイスにおいて，情報伝達に使用される転送(Informationtransfer)フェーズが示されています．

データフェーズは，データのやりとりの際のフェーズで，送信(Data Out)フェーズと受信(Data In)フェーズがあります．

コマンドフェーズは，データ転送要求などのコマンドを送信するのに使われます．

ステータスフェーズは，実行結果のステータス(正常終了00H)を返す時点で利用されます．

メッセージフェーズには，送信(Message Out)フェーズと受信(Message In)フェーズがあり，前者はコマンドのアボート(中断打ち切り)，後者はコントローラがバスから切り離しを宣言するときなどに使用されます．

なお，コマンドなどはデバイスに依存する点が多いため，詳細はデバイスの説明書を参照してください．

7.7 RS-232C インタフェース

FMTOWNSには，RS-232規格に沿った，シリアルインタフェースが用意されています．RS-232Cインタフェースの制御には，8251相当のUSART(Universal Synchronous Asynchronous Reciever/Transmitter)と呼ばれるコントローラが用いられています．以下，このモジュールを単に「8251」と呼ぶことにします．

この節では，RS-232Cインタフェースの仕組みと働きについて解説します．

7.7.1 RS-232Cコネクタと内蔵モデムのコネクタ

FMTOWNSには，RS-232Cインタフェースのコネクタが2系統あります．

1つは，通常のRS-232C仕様のコネクタで，もう1つは内蔵型のモデムを接続するコネクタです．一度には，どちらか一方しか使用できません．

RS-232Cコネクタに対する扱いと，内蔵モデムに対する扱いは基本的に同じで，後述のモデム制御レジスタ(表I-7-43)への書き込みによって，どちらを使用するかを選択することができます．なお，内蔵モデムは非同期式通信用のモデムのため，同期式通信の設定をしても正しく動作しません．

7.7.2 RS-232Cコントローラの仕様

RS-232Cコントローラの仕様は表I-7-34のようになっています．

▼表 I-7-34　RS-232Cコントローラの仕様

項　目	仕　　　様
コントローラ	8251(調歩/同期)相当
ボーレート	300/600/1200/2400/4800/9600/19200bps (ボーレートの設定はタイマにて行う)
信号線	SD, RD, RS, CS, DR, ER, CI, ST1, ST2 RT, CD
割り込み要因	TXRDY, RXRDY, SYNDET, CI-ON, CS-ON 個別にマスク可能

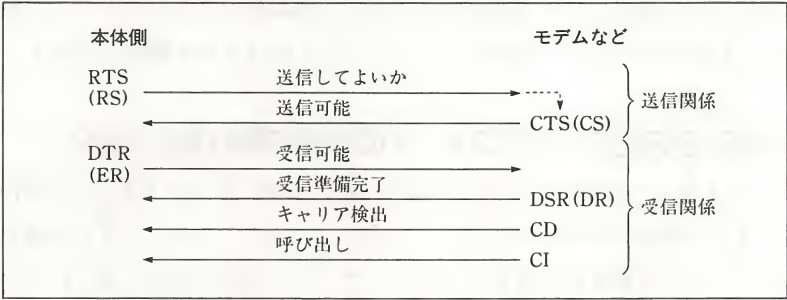
7.7.3 RS-232Cインタフェースの信号線とその働き

RS-232Cコネクタには，送受信データとハンドシェークのための信号線が接続されており，コンピュータ(8251を含む)と外部のデバイスとのやり取りを行います．主要な信号線の働きを図 I-7-19に示します．

▼図 I-7-19　RS-232Cインタフェースの主な信号線

本体側　　　　　　　　　　　　　　　　モデムなど

RTS (RS) → 送信してよいか →
← 送信可能 ← CTS(CS)　} 送信関係

DTR (ER) → 受信可能 →
← 受信準備完了 ← DSR(DR)
← キャリア検出 ← CD
← 呼び出し ← CI　} 受信関係

● FG

FGは，フレームグランド，つまりアースです．

● SG(GND)

SGはシグナルグランド線です．他の信号線は，この線との電圧レベルの差で値が0であるか1であるかを示します．

● RD(Receive Data)，SD(Send Data)

それぞれ，受信データ線，送信データ線です．シリアル変換されたデータはこのデータ線を通して送受信されます．

● RS(Request To Send)/CS(Clear To Send)

この2つは出力に関係するハンドシェークラインです．RSは，自分が送信準備ができていることを相手に示す出力線で，CSは，相手が送信を許可したことを自分が知るための入力線です．

● DR(Data Set Ready)/ER(Data Terminal Ready)

この2つは，入力に関係するハンドシェークラインです．DRは，相手側が送信可能状態にあることを知るための入力線であり，ERは自分が受信可能であることを相手に示す出力線です．

● CD(Carrier Detect)

モデムが相手先のモデムと回線上で接続されており，送受信可能な状態であることを示す線です．回線が切断された場合は，その時点で，CD信号も落ちます．

● CI(Calling Indicator)

モデムが接続されている回線から，接続要求がきていることを示します．電話でいうならば，呼び出し音が鳴っている状態をさします．

RS-232Cには，他にも信号線がありますが，FMTOWNSでは，ユーザーが制御できるのは，これらの信号線のみとなっています．以上の信号線はFMTOWNSが親，接続先が子とみなしているので，そのまま他のパソコンと接続するとお互いが親として働き衝突します．これを避けるには，クロスパッチケーブルを使用しなければなりません．

また，内蔵モデムの場合は，出力端子がそのまま，電話線のコネクタとなっており，上記のポートに出力されているピンが，内蔵モデムにそのまま結線される形になります．

7.7.4 RS-232Cインタフェースの制御に関わるレジスタ

RS-232Cインタフェースのコントローラには，8251が使われていますが，その内部には，受信データレジスタ，送信データレジスタ，コマンドレジスタ，ステータスレジスタ，モードレジスタがあり，これらに値を書き込むことによって，データ転送を行います．なお，割り込み制御，ボーレートの設定などのために，8251内部以外に他のレジスタがあります．

以下にそれらのレジスタについて，実際に，データ転送を行う場合を想定して解説をします．

●モードレジスタ

RS-232Cインタフェースを使用する前に，まず，8251を同期モード，非同期モードのどちらで使用するかの選択と，データ転送のプロトコル(取り決め)を設定します．

同期モードと非同期モードは，データ転送上，重要な形態なので，ここで少し説明します．

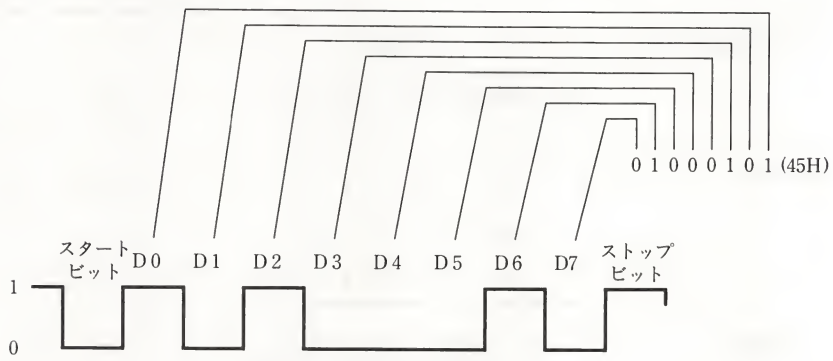
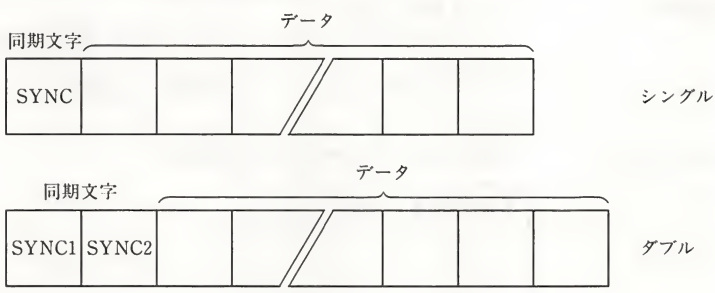
●非同期モードと同期モード

シリアル転送においては，データは決められた時間間隔で1ビットのON，OFFの信号が連続しているだけのものなので，続いている信号のどこからどこまでが1キャラクタであるかということを明確にする必要があります．

非同期式は，1キャラクタずつ同期をとることを特徴としています．すなわち，キャラクタのデータをスタートビット(0)とストップビット(1)で挟むようにします．図I-7-20に，非同

期式の信号の状態を示します．このようにするとストップビットが確実に受け取れているかどうかで，正しくデータが取り込めているかどうかを判断することができます．ただし，その位置がデータビットで１だった場合は，誤りを見逃がしてしまいます．

同期式は，一定のビットパターンの同期文字(SYNC キャラクタ)を決めておき，この文字が正しく復元されるビット位置に合わせて，ビットの組み立てを行うものです．SYNC キャラクタには，１バイト(シングル)と２バイト(ダブル)の場合があります(図 I-7-21)．なお，一般のパソコン通信は，非同期式で行われています．

▼図 I-7-20　非同期式でデータ45Hを送信した場合の信号

▼図 I-7-21　同期式の同期文字とデータの並び方

●モードレジスタの設定

モードの設定には，モードレジスタを使用しますが，モードの違いによって，各ビットの意味が異なるので，それぞれを同期モードレジスタ，非同期モードレジスタの２つに区別して考えることができます．また，各モードレジスタとコマンドレジスタの書き込みには，ともに，I/O アドレス 0A02H 番地を使います．このアドレスは，システムの起動時にはモードレジスタになっており，モードレジスタに対する書き込みが終了すると，コマンドレジスタのアドレスとなります．

なお，コマンドレジスタのIRビットに1をセットすると，8251がインターナルリセット状態となり，I/Oアドレス0A02H番地は，モードレジスタとして使用できるようになります．

●非同期モードレジスタ

非同期モードレジスタ(表I-7-35)には，ストップビット長，キャラクタ長，パリティ，送受信クロックの分周比などを設定します．

▼表I-7-35　非同期モードレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A02H	非同期モードレジスタ	W	S2	S1	EP	PEN	L2	L1	B2	B1

S2，S1(bit7，6)　：ストップビット長指定．

ストップビット長	S2	S1
禁　止	0	0
1 bit	0	1
1.5bit	1	0
2 bit	1	1

EP(bit5)　：パリティの偶数，奇数の指定．
0＝奇数(ODD)パリティの指定
1＝偶数(EVEN)パリティの指定
PEN(bit4)が0のときには意味を持たない．

PEN(bit4)　：パリティ有無の指定．
0＝受信時パリティチェックを行わない．送信時にはパリティビットを付加しない
1＝受信時パリティチェックを行う．送信時にはパリティビットを付加する

L1，L2(bit2，3)　：送受信するデータのキャラクタ長指定．

キャラクタ長	L2	L1
8 bit	1	1
7 bit	1	0
6 bit	0	1
5 bit	0	0

B2，B1(bit1，0)　：送受信クロックの分周比の指定．

分周比	B2	B1
同期	0	0
1/1	0	1
1/16	1	0
1/64	1	1

B1＝0，B2＝0のときは同期モードの設定とみなされる．

1/1のときは外部同期のときのみ可能．
(FMTOWNSでは不可)

ストップビット長は，ストップビットの時間的長さを示します．キャラクタ長はキャラクタのコードのビット数を示します．

分周比は，8251に対して与えられるクロックを何分周したものを，ボーレートクロックとして使うのかを指定します．データ取り込みのタイミングのマージンが，この値に影響を受けます．これをレシーブマージンと呼んでいます．分周比が大きいほど，つまり，入力クロックとボーレートクロックの比が大きいほど，レシーブマージンが大きくなり，エラーが少なく受信できるようになります．分周比とボーレートの関係については，第3章の表I-3-26を参照してください．

●同期モードレジスタ

同期モードレジスタ(表I-7-36)には，キャラクタ長，パリティ，SYNCキャラクタの長さを指定します．

▼表I-7-36　同期モードレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A02H	同期モードレジスタ	W	SCS	ESD	EP	PEN	L2	L1	0	0

SCS(bit7)　：SYNCキャラクタモードの指定．
　0＝ダブルSYNCキャラクタモード
　1＝シングルSYNCキャラクタモード

ESD(bit6)　：同期モードの指定．
本来は内部同期モード，外部同期モードの切り換えに用いるが，外部同期モードを使用することができないので，常にこのビットは0にして内部同期モードのみ選択する．

EP(bit5)　：パリティの偶数，奇数の指定．
　0＝奇数(ODD)パリティの指定
　1＝偶数(EVEN)パリティの指定
PEN(bit4)が0のときには意味を持たない．

PEN(bit4)　：パリティ有無の指定．
　0＝受信時パリティチェックを行わない．送信時にはパリティビットを付加しない
　1＝受信時パリティチェックを行う．送信時にはパリティビットを付加する

L2，1(bit3，2)　：送受信するデータのキャラクタ長指定．

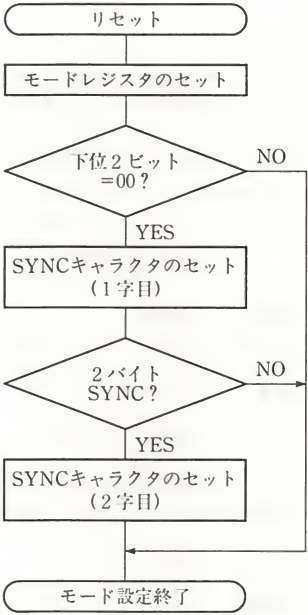
キャラクタ長	L2	L1
8 bit	1	1
7 bit	1	0
6 bit	0	1
5 bit	0	0

SCSは，SYNCキャラクタを1キャラクタにするか，2キャラクタにするかを設定します．

SCSを除いた各々の意味は，非同期モードとまったく同じです．

通信モードの設定を書き込んだ後に，SYNCキャラクタが1キャラクタの場合は1バイト，2キャラクタの場合は2バイトのSYNCキャラクタの値を続けて書き込みます．そのアルゴリズムを図I-7-22に示します．

モードレジスタに対して書き込みが終わると，コマンドレジスタによって，インターナルリセットしない限り，モードレジスタに書き込むことはできません．また，前に8251を使用していたプログラムが，同期通信モードの初期化を行っている最中に，処理を中断している場合もありえます．その際に，コマンドレジスタに対する書き込みを保証するために，I/Oアドレス0A02H番地に，3回0を書き込んだ後に，インターナルリセットを実行します．

▼図I-7-22　モードレジスタの操作手順

●コマンドレジスタ

コマンドレジスタ(表I-7-37)は，受信可能や送信可能を通知したり，ブレークキャラクタ(全ビット0)送出の指示などを指定します．

EHは，同期モードでSCYNキャラクタを捜すための制御を行います．

IRは，8251の初期化のためにインターナルリセット状態にするために使われます．

RTS，DTRは，信号線RS/ERに出力するために使われます．

ERRRSTは，1をセットすることによって，後述のPE，OE，FEの各エラー状態フラグをクリアするために使われます．

SBRK は，ブレークキャラクタを送出します．

RxEN は，8251の受信回路を受信可能状態に設定します．TxE ビットは，8251の送信回路を送信可能状態にします．これらのビットが０の状態では，受信も送信もできません．

モードレジスタの設定が終った後，送受信するためには，コマンドレジスタの RTS，DTR ビットをセットして，送信要求，受信許可をし，さらに RxEN，TxE ビットをセットして，送受信可能な状態にします．割り込み制御にする場合は，ユーザーがアプリケーションで，割り込みを行うようにプログラムする必要があります．

▼表 I-7-37　コマンドレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A02H	コマンドレジスタ	W	EH	IR	RTS	ERR RST	SBRK	RxEN	DTR	TxE

EH (bit7)　：ハントモードの実行(このビットは同期モードのみ意味を持つ)．
このビットを１にするとSYNCキャラクタのハントモードに入り，SYNCキャラクタをサーチする．SYNCキャラクタが見つかったときにはSYNDETを１にしてハントモードを終了する．
このビットはハントモード終了後，自動的に０にもどる．
SYNDETはステータスレジスタ(0A02H)を読み出すことによって０にもどる．

IR (bit6)　：インターナル(内部)リセット．
１＝インターナル(内部)リセット
IR後モードレジスタ設定に入る．

RTS (bit5)　：RS信号の制御．
０＝RS信号OFF
１＝RS信号ON

ERRRST (bit4)　：ステータスレジスタのPE(パリティエラー)，OE(オーバーランエラー)，FE(フレームエラー)の各フラグのクリア．
１＝各フラグのクリア
０にもどす必要はない．

SBRK (bit3)　：ブレーク信号(ブレークキャラクタ)の送信の制御．
０＝ブレーク送出停止
１＝ブレーク送出

RxEN (bit2)　：受信イネーブル．
０＝ディスエーブル
１＝イネーブル

DTR (bit1)　：ER信号の制御
０＝ER信号OFF
１＝ER信号ON

TxE (bit0)　：送信イネーブル．
０＝ディスエーブル
１＝イネーブル
データの送信はTxEとCSが両方１のとき，行われる．

●ステータスレジスタ1

ステータスレジスタ1(表I-7-38)は，データ転送のステータスをチェックするものです．データ転送の際には，このレジスタをチェックしながら，送信データレジスタや，受信データレジスタをアクセスする必要があります．

ステータスレジスタ1では，現在の送信／受信データレジスタの状態，通信エラーの状態などを知ることができます．

DSRは，現在のDR信号線の状態をそのまま反映します．

SYNDET/BDは，非同期モードではブレークキャラクタを検出したとき，同期モードでは，SYNCキャラクタを受け取ったときに1となります．

FEが1であるとき，フレーミングエラーであることを示します．フレーミングエラーは，調歩通信時に，ストップビットが正しく検出されなかったときに発生します．

OEが1であるとき，オーバーランエラーが起きたことを示します．オーバーランエラーは，受信データが受信データレジスタから読み出される前に，次のデータが受信され，前の受信データがなくなってしまった場合に起きます．

PEが1であるとき，パリティエラーが起きたことを示します．パリティエラーは，モードレジスタに書き込まれたパリティチェックを受信データに対して行ったときに，エラーとなった場合に起きます．

FE，OE，PEが1になると，8251の該当ビットは，ステータスレジスタを読み出しただけでは，リセットしません．これらのビットをリセットするには，コマンドレジスタのERRRSTを1にします．

TxEは，送信レジスタの内容が空で，かつ，送信中のデータがないことを示します．

RxRDYは，受信データが送信レジスタに格納されて，読み出し可能になっていることを示します．

TxRDYは，送信データレジスタが空で，次のデータを受け付けることができる状態であることを示します．TxEと異なるのは，送信レジスタが単に空であることを示している点です．送信レジスタは2段構成となっており，後段は並列→直列変換を行うシフトレジスタが配置されています．シフトレジスタが空になると前段の内容が転送され，その結果，前段のレジスタは空になります．これがTxRDY＝1の状態です．シフトレジスタのデータがなくなり，前段も空のままではTxE＝1となります．送信データが書き終わり，8251の送信動作が完全に終了したことを確認するにはTxEを参照します．

▼表 I-7-38　ステータスレジスタ1

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A02H	ステータスレジスタ1	R	DSR	SYNDET /BD	FE	OE	PE	TxE	RxRDY	TxRDY

DSR(bit7)　：RS-232C回線のDR信号線の状態を示す.
　0＝DR信号線OFF
　1＝DR信号線ON

SYNDET/BD(bit6)　：非同期モード時はブレーク信号(ブレークキャラクタ)の検出.
同期モード時はSYNCキャラクタの検出に用いる.
　0＝非検出
　1＝検出
SYNDET信号はステータス読み出し動作により自動的にリセットされる.
ブレーク信号はマスタリセットがかかった時または, Rxデータが1になったときリセットされる.

FE(bit5)　：フレーミングエラー.
　0＝FEなし
　1＝FE発生
FEを解除するにはコマンドレジスタ(0A02H)のERRRST(bit4)を1にする.

OE(bit4)　：オーバーランエラー.
　0＝OEなし
　1＝OE発生
OEを解除するにはコマンドレジスタ(0A02H)のERRRST(bit4)を1にする.

PE(bit3)　：パリティエラー.
　0＝PEなし
　1＝PE発生
PEを解除するにはコマンドレジスタ(0A02H)のERRRST(bit4)を1にする.
受信したデータのパリティチェックはモードレジスタ(0A02H)のbit4を1にしたとき行われる.

TxE(bit2)　：送信データレジスタの状態を示す.
　0＝送信データあり
　1＝送信データエンプティ
　　非同期モード：データが書き込まれるまで，アイドル状態になる.
　　同期モード：データが書き込まれるまで，SYNCキャラクタ送信
送信レジスタにデータを書くことによりTxEは自動的に0になる. TxEは送信の終了を示すのに使用でき，半二重モードでいつラインを反転させるか知ることができる.

RxRDY(bit1)　：受信データレジスタの状態を示す.
　0＝受信データなし. もしくは，読み込み中
　1＝受信データあり(受信データ読み出し可)
このビットはブレーク状態でもセットされる.
コマンドレジスタのRxEN(bit2)が0のときは，セットされない.
受信データレジスタからデータを読み出すことによってRxRDYは自動的に0になる.

TxRDY(bit0)　：送信データレジスタの状態を示す.
　0＝送信データあり
　1＝送信データエンプティ
送信データレジスタにデータを書き込むことによってTxRDYは自動的に0になる.
このビットは，TxRDY端子とは異なり，TxE，CTS端子の状態には依存しない.
TxRDY　　＝1(データバッファが空)
TxRDY端子＝H(データバッファが空)・(＝CTS＝L)・(TxE＝1)

●データレジスタ

データレジスタ(表I-7-39)は，読み出すことによって受信データを取り出すことができ，書き込みによって送信ができます．読み出す場合を受信データレジスタ，書き込む場合を送信データレジスタといいます．

実際の送信は，ステータスレジスタ1のTxRDYビットが1になるのを待ち，送信データレジスタにデータを書き込むことによって行われます．また，受信はステータスレジスタのRxRDYビットが1になるのを待ち，受信データレジスタから読み出せばよいのですが，現実には，受信はどのタイミングで起きるか分らず，受信したデータをCPUで転送する間に，次のデータがきてデータが失われてしまうことがあるので，割り込み処理にするのが一般的です．送信の時間待ちにも，割り込みを使います．

▼表I-7-39　データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A00H	受信データレジスタ	R	D7	D6	D5	D4	D3	D2	D1	D0
	送信データレジスタ	W								

レジスタ長は8ビットであり，8ビットに満たないデータ長のときにはレジスタのD0側から右づめで読み書きする．残りのビットは0．

●ステータスレジスタ2

ステータスレジスタ2(表I-7-40)は，RS-232Cの信号線のDR，CD，CS，CIの状態を見ることができます．上記のとおりのビット構成となっています．

▼表I-7-40　ステータスレジスタ2

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A04H	ステータスレジスタ2	R	不定				DSR	CD	CS	CI

DSR(bit3)　：RS-232C回線のDR信号線の状態を示す．
　0＝DR信号線OFF
　1＝DR信号線ON
　このビットはステータスレジスタ1のDSR(bit7)と同じである．

CD(bit2)　：RS-232C回線のCD信号線の状態を示す．
　0＝CD信号線OFF
　1＝CD信号線ON

CS(bit1)　：RS-232C回線のCS信号線の状態を示す．
　0＝CS信号線OFF
　1＝CS信号線ON

CI(bit0)　：RS-232C回線のCI信号線の状態を示す．
　0＝CI信号線OFF
　1＝CI信号線ON

●割り込み要因レジスタ

割り込み要因レジスタ(表I-7-41)は，RS-232Cインタフェースの制御においては，いつ発生するか分らない要因や時間待ちに対しては，割り込みによって対応しています．このレジスタは，シリアルインタフェースのどの部分が，CPUに対して割り込みをかけたのかを示します．CI, CSは，RS-232Cの信号線CI, CSがアクティブになったことによって割り込みがかかったことを示し，RSINTは，TxRDY, RxRDY, SYNDETのいずれかがアクティブになったことによって，割り込みが起きたことを示します．なお，TxRDY，RxRDY，SYNDETの，どれがアクティブになったのかは，ステータスレジスタ1を読み出すことによって調べることができます．

▼表I-7-41　割り込み要因レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A06H	割り込み要因レジスタ	R	不　定					CI	CS	RSINT

CI(bit2)　　：CIによる割り込みを示す．
　0＝割り込みなし
　1＝割り込みあり

CS(bit1)　　：CSによる割り込みを示す．
　0＝割り込みなし
　1＝割り込みあり

RSINT(bit0)　　：TxRDY，RxRDY，SYNDETのいずれかによる割り込みを示す．
　0＝割り込みなし
　1＝割り込みあり

●割り込み制御／クロック切り換えレジスタ

割り込み制御／クロック切り換えレジスタ(表I-7-42)は，送信受信クロックを外部から取るのか内部のクロックを用いるのかの指定をします．DR信号線の制御，割り込みのマスクの指定ができます．

●モデム制御レジスタ

モデム制御レジスタ(表I-7-43)は，内蔵モデムを使うのか，それとも外部のRS-232Cインタフェースを用いるのかを設定します．

なお，モデムカードを挿入すると，自動的に内蔵モデムの側となります．このレジスタはこの状態で外部のRS-232Cインタフェースに切り換えるときなどに使用します．

▼表 I-7-42　割り込み制御/クロック切り換えレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A08H	割り込み制御/クロック切り換えレジスタ	W	TxC	RxC	EXT DTR	CI	CS	SYN DET	RxRDY	TxRDY

TxC (bit7)　：送信タイミング信号の選択.
0＝内部クロック
1＝外部クロック

RxC (bit6)　：受信タイミング信号の選択.
0＝内部クロック
1＝外部クロック

EXTDTR (bit5)　：ER信号線の制御.
0＝ER信号線の状態はコマンドレジスタ(00A2H)のビット1のDTRの状態に依存する
1＝ER信号線をONにする

CI (bit4)　：CI信号ONによる割り込み制御.
0＝割り込み禁止
1＝割り込み許可

CS (bit3)　：CS信号ONによる割り込み制御.
0＝割り込み禁止
1＝割り込み許可

SYNDET (bit2)　：SYNDET信号ONによる割り込み制御.
0＝割り込み禁止
1＝割り込み許可

RxRDY (bit1)　：RxRDY信号ONによる割り込み制御.
0＝割り込み禁止
1＝割り込み許可

TxRDY (bit0)　：TxRDY信号ONによる割り込み制御.
0＝割り込み禁止
1＝割り込み許可

▼表 I-7-43　モデム制御レジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A0AH	モデム制御レジスタ	R	ENBL	MODEM	MODINS	不定				
		W			0	0	0	0	0	0

ENBL (bit7)　：　ハード/ソフト切り換えビットを示す.
0＝ハードウェア切り換え
1＝ソフトウェア切り換え

MODEM (bit6)　：　RS-232Cまたは内部モデムの選択を示す.
0＝RS-232C
1＝内蔵モデム

MODINS (bit5)　：　モデムカードの有無を示す.
0＝モデムカードあり
1＝モデムカードなし

ハードウェア切り換えは，モデムカード抜き差しにより切り換える．ソフトウェア切り換えは，MODEM (bit6) により切り換える.

第II部

FM TOWNSのBIOS

BIOS の概要

グラフィック BIOS

スプライト BIOS

マウス BIOS

フォント BIOS

サウンド BIOS

CD-ROM BIOS

キーボード BIOS

ディスク BIOS

プリンタ BIOS

時計をサポートする BIOS

RS-232C BIOS

ブザーBIOS

割り込み管理 BIOS

サービスルーチンと拡張サービスルーチン

システム情報 BIOS

音源割り込み管理 BIOS

MIDI マネージャBIOS

第 1 章

BIOSの概要

FMTOWNSでは，I/Oの入出力処理をBIOS(Basic Input Output System)と呼ばれるI/Oドライバモジュールで行います．BIOSは，FMTOWNSのハードウェア機能を論理的に扱うことを可能にし，FMTOWNS上で動作するプログラムに共通のソフトウェアインタフェースを提供しています．

この章では，FMTOWNSで使用できるBIOSの種類と，BIOSを使う際に知っておくべき基本的な手順について解説します．個々のBIOSについては次章以降を参照してください．

1.1 FMTOWNSのBIOS

FMTOWNSには，次の表に示すように，現在，2とおりの動作環境が用意されています．その動作環境に応じて，BIOSも2種類に大別できます．2種類のBIOSの概要を表II-1-1に示します．

BIOSは，ハードウェアとソフトウェアのインタフェースモジュールであり，TownsOSや日本語MS-DOSは，このBIOSの上に構築されます．BIOSのモジュールは，OSのシステムソフトとして外部記憶装置(フッピィディスクや，CD-ROM)に収められており，システムの起動時に読み込まれるようになっています．この第II部では，FMTOWNSの独自の動作環境を実現しているBIOS(TownsOS Ver. 2.1Level 31システムのBIOS)について解説を行います．

▼表II-1-1　FMTOWNSの2種類の動作環境とBIOS

OSの種類	BIOS	動作環境
Towns OS (MS-DOS+386| DOS-Extender)	FMTOWNS独自のBIOS (FMR-50と共通のBIOSを含む)	FMTOWNSの独自の動作環境 ・80386コードの実行 ・FMTOWNS独自の制御(AVなど)
日本語MS-DOS	FMR-50互換のBIOS	FMR-50と同等の動作環境 ・FMR-50のアプリケーションの実行

なお，日本語MS-DOSのBIOSは，ユーザー側から見た場合，基本的にFMR-50と同等になっているので，本書では解説しません．詳しくはFMR-50のBIOS解説書に従ってください．また，FMTOWNSでFMR-50互換のプログラムを作る場合の注意点などは，FMTOWNSのMS-DOSに関するマニュアルなどを参照してください．

1.2　TownsOS上で使用できる2系統のBIOS

TownsOSのもとで使用できるBIOSは，表II-1-2に示すようにFMTOWNSで新規に開発されたネイティブBIOSと，FMRシリーズのBIOSから引き継がれたリアルBIOSの2系統に分けられます．

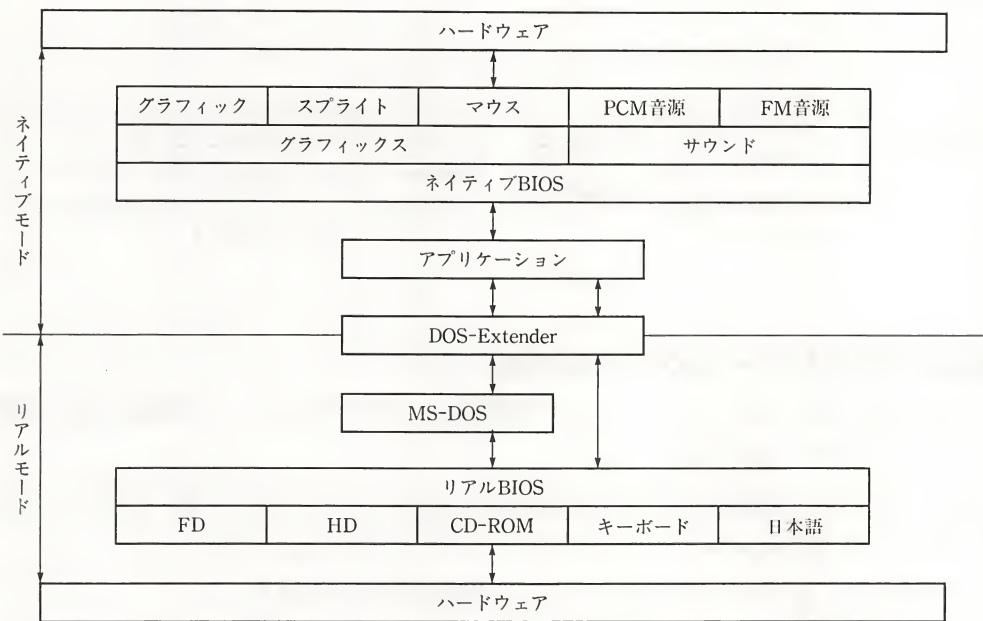
この第II部では，ネイティブBIOSと，リアルBIOSのうち仕様が大幅に拡張されたCD-ROM BIOSについて詳しく解説します．その他のリアルBIOSについては，これまで，他の書籍などで触れられているので，第8章以降で簡単に解説するにとどめます．

▼表II-1-2　TownsOS上で使用できるBIOS

ネイティブBIOS	リアルBIOS
グラフィック	CD-ROM
スプライト	キーボード入力
マウス	補助記憶装置(フロッピィディスク，ハードディスク)
フォント	プリンタ
サウンド	時計(カレンダ時計，タイマ管理，時計管理)
システム情報	RS-232C
音源割り込み管理	ブザー
MIDIマネージャ	割り込み管理
	拡張サービスルーチン
	各種サービスルーチン

1.3　BIOSとハードウェアの関係

TownsOSのBIOSとハードウェアの関係を図II-1-1に示します.

▼図II-1-1　Towns OSのBIOSとハードウェアの関係

1.4　TownsOS上のプログラム実行環境と80386のプログラムの実行

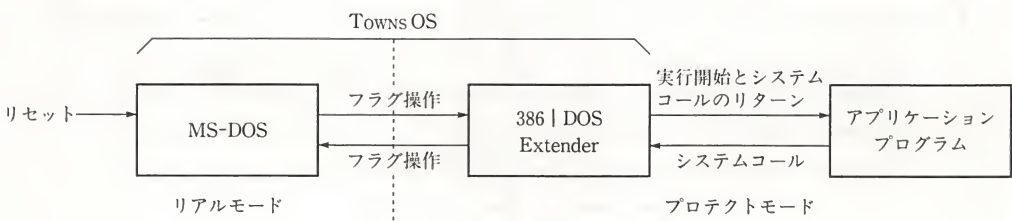
TownsOSは, MS-DOSに386 | DOS-Extender™(以下, DOS-Extenderと表記する)を付属させたものです. DOS-Extenderを使うことにより, MS-DOSの配下で, 80386のプロテクトモード(ネイティブモード)のプログラムを実行する(32ビットのレジスタを使用したり, 1MBを超えるメモリをアクセスしたりする)ことができるようになります.

● MS-DOSとDOS-Extenderの関係

プログラムで, MS-DOSおよび, BIOSのシステムコールを行う際には, 特別な方法ではなく, ソフトウェア割り込みを使った標準的な方法を用います.

DOS-Extender は，リアルモード(MS-DOS)と386ネイティブモード(アプリケーション)間でのデータのやり取りを自動的に行うので，プログラムの大きさやメモリ構成に関係なく，MS-DOS の機能を利用することができるようになります．

TownsOS 上のアプリケーションプログラムの実行環境を図II-1-2に示します．

▼図II-1-2　Towns OS 上のアプリケーション実行環境

●80386のプロテクトモードのプログラムの実行

TownsOS 上では，386プロテクトモードで動作するプログラムファイルの拡張子は〝.EXP″にするという決まりがあります．

このプログラムを実行するには，TownsMENU 上で，ファイル名をマウスでクリックします．すると，DOS-Extender によってプログラムがロードされます．

この段階では，各レジスタには表II-1-3に示す値がセットされます．

そして，プログラムに制御が移ります．なお，DOS-Extender は，CPU フラグの切り換えによって，プログラムをプロテクトモード(ネイティブモード)で実行します．

▼表II-1-3　レジスタの初期値

レジスタ	初期値
DS	0014H
ES	0014H
FS	0014H
GS	0014H
CS	000CH
EIP	プログラムエントリアドレス
SS	0014H
ESP	スタックの先頭のアドレス
EAX	0000000H
EBX	0000000H
ECX	0000000H
EDX	0000000H
ESI	0000000H
EDI	0000000H
EBP	0000000H

1.5　BIOSを使用するための手順

ここでは，TownsOS上のBIOSを使用するための手順とBIOSコールのサンプルを紹介します．

● BIOS呼び出しの手順

BIOSを使用する手順は，FMTOWNS独自のBIOS(ネイティブBIOS)の場合も，FMRシリーズと共通のBIOS(リアルBIOS)の場合も基本的に同じですが，ネイティブBIOSは，far CALL命令で呼び，一方，リアルBIOSはINT命令で呼ぶ点が異なります．

BIOS呼び出しの手順を説明します．

①エントリ(BIOSの呼び出しのとき)

機能コード

AHレジスタに設定

データパラメータ

1バイトのデータは，ALレジスタ．

1バイト以外のデータは，DX，BX，CXレジスタ．

ネイティブBIOSでは，この外，ECXなどの32ビットのレジスタも使用する．

アドレスパラメータ

オフセットアドレスをDIレジスタ，セグメントアドレスをDSレジスタ．

アドレスの表記法はDS:DIとする．

ネイティブBIOSでは，DI:ESIなども使用する．

② BIOSの呼び出し

ネイティブBIOSの場合は，far CALL命令で行う．

リアルBIOSの場合は，INT命令を使用する．

③リターン(BIOSから復帰したとき)

エラーフラグ

キャリーフラグ(フラグレジスタのCFビット)に設定　(0：正常　1：エラー)

エラーコード

[リアルBIOSの場合]

AHレジスタに設定

00H	：正常
01H	：未定義機能コード
02H～7FH	：エラーコード
80H	：拡張エラーコード(詳細情報は，CXレジスタ)

［ネイティブBIOSの場合］

AHレジスタに設定

00H　　　　　：正常

00H以外　　　：エラー

グラフィックBIOS，スプライトBIOS，マウスBIOS，フォントBIOSは，エラーの場合，FFHが返る．また，サウンドBIOSでは，ALにエラーの種類が返る．

データパラメータ

AL，DX，BX，CXレジスタに設定

ネイティブBIOSでは，SI，DIなどの16ビットレジスタやECXなどの32ビットレジスタを使用する．

アドレスパラメータ

オフセットアドレスをDIレジスタ，セグメントアドレスをDSレジスタに設定する．レジスタの示すアドレスの表記法は，DS:DIとする．

ネイティブBIOSでは，DS:ESIも使用する．

④保存レジスタ

フラグ

キャリーフラグを除くフラグレジスタの値は保存．なお，ネイティブBIOSではキャリーフラグも保存．

レジスタ

BIOSから復帰したとき，パラメータが設定されるレジスタ以外のレジスタの値は，基本的に保存．

●ネイティブBIOSコールの実例

例として，グラフィックBIOSを初期化するプログラムを次に示します．

```
mov   ah, 0h                            機能コードの設定 ┐レジスタ値の
mov   edi, offset work                  ワーク領域の設定 ┘セット例
push  0110h                             BIOSの領域のセレクタの設定 ┐ BIOS
pop   fs                                                        │ コール
call  pword ptr fs:〔入口アドレス〕     呼び出し                  ┘
```

DOS-ExtenderにBIOS領域のセレクタを与えるために，FSレジスタに0110Hをセットして，入口アドレスをコールします．入口アドレスは，各BIOSによって固有の値を取ります．表II-1-4は，その値を示したものです．

▼表II-1-4　ネイティブBIOSの種類と入口

BIOSの種類	入口アドレスの値
グラフィックBIOS	20H
マウスBIOS	40H
スプライトBIOS	60H
サウンドBIOS	80H
フォントBIOS	A0H
MIDIマネージャBIOS	C0H
音源割り込み管理BIOS	1A0H
システム情報BIOS	1C0H

●リアルBIOSコールの実例

普通のMS-DOSのファンクションコールの場合と同様に，AHレジスタに機能コードを入れておき，INT番号で呼びます．

ただし，リアルBIOSは，リアルモードで動作するので，エントリやリターンのデータ領域(パラメータの領域)として，ユーザーメモリの1MB以内の範囲しか指定できません．そこで，ネイティブモードのプログラム中で，パラメータの値を1MBを超えるメモリ空間へ読み出したり，書き込んだりする必要がある場合は，リアルモードとネイティブモードの切り換えを行って，1MB以内のメモリと1MBを越えるメモリの間でデータ転送を行うようにします．

ただし，データの受け渡しが，レジスタだけで済んでしまう場合は，この必要はありません．

次ページ以降にサンプルのプログラムを示します．

なお，各BIOSによって，INT番号が異なります．表II-1-5に各BIOSのINT番号を示します．

▼表II-1-5　リアルBIOSの種類とINT番号

BIOSの種類	INT番号
CD-ROM BIOS	93H
キーボードBIOS	90H
ディスクBIOS	93H
プリンタBIOS	94H
カレンダ時計BIOS	96H
タイマ管理BIOS	97H
時計管理BIOS	98H
RS-232C BIOS	9BH
ブザーBIOS	9EH
割り込み管理BIOS	AEH
サービスルーチンBIOS	AFH
拡張サービスルーチンBIOS	8EH

注意）ネイティブBIOS，リアルBIOSをアクセスするプログラム例は，FMTOWNS用のアプリケーション開発キットを使ってアセンブルすることを前提としています．

```
;*************************************************************************
;* SAMPLE1                          サンプル
;*
;*   BIOSコールがレジスタインターフェイスの場合
;*************************************************************************
;*
;* SAMPLE1がC言語プログラム中で以下のように定義されている場合
;*
;*  int SAMPLE1(int in_prm1,int in_prm2,int out_prm1,int out_prm2);
;*
;*  /* 入力 */
;*    int in_prm1;   /* BIOSコールする際の入力パラメータ1 (→register AL) */
;*
;*    int in_prm2;   /* BIOSコールする際の入力パラメータ2 (→register BX) */
;*
;*  /* 出力 */
;*    int out_prm1; /* BIOSからの出力パラメータ1(→register CX) */
;*
;*    int out_prm2; /* BIOSからの出力パラメータ2(→register DX) */
;*
;* ライブラリのコーディングは以下のようになる。
;*
                .386p
                public  SAMPLE1
codeseg         segment use32 dword 'CODE'
                assume  cs:codeseg
                db      'SAMPLE1',7      ; '文字列',文字数の登録
SAMPLE1         proc    near

FUNCTION        equ     00               ; 対応するBIOSのファンクションNo.(→AH)
INT_NO          equ     00               ; 対応するBIOSのインタラプトNo.

#in_prm1        equ     ss:[ebp+8]       ; in_prm1のアドレス
#in_prm2        equ     ss:[ebp+12]      ; in_prm2のアドレス
#out_prm1       equ     ss:[ebp+16]      ; out_prm1のアドレス
#out_prm2       equ     ss:[ebp+20]      ; out_prm2のアドレス

                enter   0,0
                mov     ah,FUNCTION      ; ファンクションNo.を設定
                mov     al,#in_prm1      ; 入力パラメータ1を設定
                mov     bx,#in_prm2      ; 入力パラメータ2を設定
                int     INT_NO           ; BIOSコール実行
                mov     #out_prm1,cx     ; ユーザプロ領域に出力パラメータ1を設定
                mov     #out_prm2,dx     ; ユーザプロ領域に出力パラメータ2を設定
                movzx   eax,ah           ; リターンコードの設定
                leave
                ret
SAMPLE1         endp
codeseg         ends
                end
```

```
;*****************************************************************************
;* SAMPLE2                              サンプル
;*
;*   BIOSコールでアドレス指定の必要がある場合
;*****************************************************************************
;*
;* SAMPLE2がC言語プログラム中で以下のように定義されている場合
;*
;*  int SAMPLE2(int in_prm1,char *out_prm1);
;*
;*  /* 入力 */
;*    int in_prm1;          /* BIOSコールする際の入力パラメータ1
;*                             (→register AL) */
;*
;*  /* 出力 */
;*    char out_prm1[256]; /* BIOSからの出力される256バイトのパラメータの
;*                             バッファ */
;*
;* ライブラリのコーディングは以下のようになる。
;*
                .386p
pmdata          segment dword public use32 'DATA'
pmdata          ends
pmcode          segment byte public use32 'CODE'
pmcode          ends

                public  SAMPLE2

                assume  ds:pmdata
pmdata          segment
rmseg           dd      ?       ; リアル,プロテクトモード間コールデータバッ
                                ;ファ(以後,共有バッファと略す)のリアルモード
                                ;アドレスのセグメントの内容
rmoff           dd      ?       ; 共有バッファのリアルモードアドレスのオフセ
                                ;ットの内容
pmseg           dd      ?       ; 共有バッファのプロテクトモードアドレスのセ
;                               ;レクタの内容
pmoff           dd      ?       ; 共有バッファのプロテクトモードアドレスのオ
                                ;フセットの内容
pmdata          ends
;
                assume  cs:pmcode
pmcode          segment
                db      'SAMPLE2',7     ; '文字列',文字数の登録
SAMPLE2         proc    near

FUNCTION        equ     00              ; 対応するBIOSのファンクションNo.(→AH)
INT_NO          equ     00              ; 対応するBIOSのインタラプトNo.
datasize        equ     256             ; BIOSから出力されるパラメータサイズ
                                        ;                          (byte)

#in_prm1        equ     SS:[ebp+8]      ; in_prm1のアドレス
#out_prm1       equ     SS:[ebp+12]     ; out_prm1の先頭アドレスのアドレス

                enter   0,0
                cld
                push    ds               ;↑
                push    es               ;|
```

```
                push    esi                 ; プログラムで使用するレジスタの保存
                push    edi                 ; | (EAX,EDXはセーブの必要なし)
                push    ebx                 ; |
                push    ecx                 ; ↓
                call    mak_buffs           ; 共有バッファの確保(mak_buffsを参照)
                mov     ah,FUNCTION         ; ファンクション番号の設定
                mov     al,#in_prm1         ; 入力パラメータ1を設定
                mov     di,word ptr rmoff;  ↑
                push    dword ptr rmseg     ; |
                push    dword ptr rmseg     ; BIOSコールの実行(BIOS_callを参照)
                push    dword ptr INT_NO    ; |
                call    BIOS_call           ; |
                add     esp,12              ; ↓
                mov     ecx,datasize        ; 共有バッファからout_prm1へのデータ
                                            ;転送バイト数の設定
                push    ds                  ; ↑
                pop     es                  ;転送先(out_prm1)アドレスの設定
                mov     edi,#out_prm1       ; ↓
                mov     esi,dword ptr pmoff ; ↑転送元(共有バッファ)アドレス
                mov     ds,word ptr pmseg   ; ↓の設定
        rep     movsb                       ; データ転送の実行
                movsx   eax,ah              ;リターンコードの設定
                pop     ecx                 ; ↑
                pop     ebx                 ; |
                pop     edi                 ; 保存レジスタの復帰
                pop     esi                 ; |
                pop     es                  ; |
                pop     ds                  ; ↓
                leave
                ret
SAMPLE2         endp
;****************************************************************************
;* mak_buffsは共有バッファを確保する為のサブルーチンです。
;*
mak_buffs       proc    near
                mov     ax,250dh                ;Extenderファンクション No.の設定
                int     21h                     ;Extenderコールの実行
                                                ;共有バッファの確保
                mov     word ptr rmoff,bx       ;rmoffの保存
                shr     ebx,16
                mov     dword ptr rmseg,ebx ;rmsegの保存
                mov     dword ptr pmoff,edx ;pmoffの保存
                mov     ax,es
                mov     word ptr pmseg,ax       ;pmsegの保存
                ret
mak_buffs       endp
;
;****************************************************************************
;* BIOS_callはリアルモードでBIOSコールをする為のサブルーチンです.
;*
RMINT           struc
RMI_INUM        dw      ?               ; インタラプトNo.
RMI_DS          dw      ?               ; リアルモードで動作するときDSの内容
RMI_ES          dw      ?               ; リアルモードで動作するときESの内容
RMI_FS          dw      ?               ; リアルモードで動作するときFSの内容
RMI_GS          dw      ?               ; リアルモードで動作するときGSの内容
RMI_EAX         dd      ?               ; リアルモードで動作するときEAXの内容
RMI_EDX         dd      ?               ; リアルモードで動作するときEDXの内容
RMINT           ends
;
```

```
BIOS_call       proc    near
#INTNO          equ     (word ptr 8[ebp])       ;インタラプトNo.
#RMDS           equ     (word ptr 12[ebp])      ;リアルモードのDSレジスタの内容
#RMES           equ     (word ptr 16[ebp])      ;リアルモードのESレジスタの内容
#RMI            equ     (dword ptr [ebp - (size RMINT)])
                enter   Ø,Ø
                sub     esp,size RMINT          ;↑
                mov     #RMI.RMI_EAX,eax        ;|
                mov     #RMI.RMI_EDX,edx        ;   リアルモードで動作するとき
                mov     ax,#RMDS                ;| の各種レジスタの内容を設定
                mov     #RMI.RMI_DS,ax          ;|
                mov     ax,#RMES                ;|
                mov     #RMI.RMI_ES,ax          ;|
                mov     ax,#INTNO               ;↓
                mov     #RMI.RMI_INUM,ax        ; インタラプトNo.の設定
                push    ds                      ; リアルモード割り込みの発行
                mov     ax,ss                   ;↑
                mov     ds,ax                   ;|
                lea     edx,#RMI                ;  BIOSコールの実行
                mov     ax,2511h                ;|
                int     21h                     ;↓
                pop     ds
                add     esp,size RMINT
                pop     ebp
                ret
BIOS_call       endp
pmcode          ends
                end
```

注）このサンプルは，リンク時または，DOS-Extender で実行する際に，-CALLBUFSのオプションを指定する必要があります．

1.6　BIOSリファレンスの見方

BIOS の各オペレーションのリファレンスは，次のような書式で記述されています．

サポートするデバイス名	CALL 命令の入口アドレスまたは，INT 命令の番号
BIOS の機能名称	BIOS 機能コード

エントリ　BIOS の呼び出しのときの設定パラメータについて説明しています．

リターン　BIOS から復帰したときの復帰パラメータについて説明しています．

説　明　BIOS の機能を詳細に説明しています．また，パラメータの詳しい説明もここで行います．パラメータブロックについては，表II-1-6に示す形式で記述しています．

▼表II-1-6　パラメータブロックの形式

(DS : DI)

E	W	パラメータ名1
R	B	パラメータ名2

E：ユーザーが設定してBIOSに渡すパラメータ
R：BIOSが設定してユーザーに返すパラメータ
なお，ネイティブBIOSでは，E，Rの記述は省略してある．

W，B，Dなどは，データの長さを示すものです．この意味を表II-1-7に示します．

▼表II-1-7　データの長さを表す記号とその意味

記号	正規表現	意　味
B	byte	1バイト(8bit)データ 符号なし(0～255)
W	word	2バイト(16bit)データ 符号なし(0～65,535)
D	double word	4バイト(32bit)データ 符号なし(0～4,294,967,296)
SB	signed byte	1バイト(8bit)データ 符号あり(－128～127)
SW	signed word	2バイト(16bit)データ 符号あり(－32768～32767)
SD	signed double word	4バイト(32bit)データ 符号あり(－2147483648～2147483647)
BA	byte array	1バイト(8bit)データの配列
WA	word array	2バイト(16bit)データの配列
DA	double word array	4バイト(32bit)データの配列
FI	fixed point	4バイト(32bit)固定小数点データ 符号あり(－32768.0～32767.9) 31　16 15　0 signed double word 小数点位置

第 2 章

グラフィックBIOS

この章では，画面制御を行っているグラフィック BIOS について解説します．

FMTOWNS のグラフィック BIOS は，FMR シリーズのものに比べ大幅に機能が拡張されているところから，EGB : Enhanced Graphic Bios(拡張グラフィック BIOS)と呼ばれています．

これらの BIOS を使用する際には，最初に「初期化(機能コード 00H)」，または「仮想画面の設定(機能コード 01H)」と「書き込みページの指定(機能コード 05H)」の両方を行い，ハードウェアを初期化することが必要です．

2.1 グラフィック BIOS 一覧

グラフィック BIOS は次の 4 種類に分類することができます．

1．セッティングオペレーション(表II-2-1)

EGB に関するハードウェアのパラメータを設定する機能です．

2．ブロックオペレーション(表II-2-2)

画面上の矩形域内のイメージデータをメモリへ保存したり，その保存したデータを再度表示する．また，回転やぼかしを行うといった機能です．

3．グラフィックオペレーション(表II-2-3)

図形の大きさや位置を指定して描画する機能です．

4．フォントオペレーション(表II-2-4)

文字の表示に関する機能です．

▼表II-2-1　セッティングオペレーション

機能名称	機能コード	機能名称	機能コード
初期化	00H	画面マスクの設定	10H
仮想画面の設定	01H	ペンの設定	11H
表示開始位置の設定	02H	ペンの太さの設定	12H
ビューポートの設定	03H	ペンの形状の設定	13H
パレットレジスタの設定	04H	マスクビットの設定	14H
書き込みページの指定	05H	文字方向の設定	15H
表示ページの設定	06H	文字表示方向の設定	16H
描画色の設定	07H	文字間空白の設定	17H
描画色の設定1	08H	文字拡大率の設定	18H
混色比率の設定	09H	字体の設定	19H
描画モードの設定	0AH	スーパーインポーズの設定	1AH
線分パターンの設定	0BH	デジタイズの設定	1BH
面塗りモードの設定	0CH	解像度ハンドルによる仮想画面の設定	1CH
ハッチングパターンの設定	0DH	グラフィック描画スタック領域の動的変更	1DH
タイルパターンの設定	0EH	デジタイズ画面取り込み位置の補正	1EH
画面マスク領域の設定	0FH		

▼表II-2-2　ブロックオペレーション

機能名称	機能コード
全画面の消去	20H
画面の消去	21H
ドットデータの読み出し	22H
ドットデータの書き込み	23H
ドットデータの読み出し1	24H
ドットデータの書き込み1	25H
ドットデータの読み出し2	26H
ドットデータの書き込み2	27H
グラフィックカーソル	28H
マスクデータの書き込み	29H
全画面スクロール	2AH
部分画面スクロール	2BH
領域の設定	2CH
画面の複写	2DH
画面の回転	2EH
画面ぼかし	2FH

▼表II-2-3　グラフィックオペレーション

機能名称	機能コード
ポイント	40H
連続線分	41H
不連続線分	42H
多角形	43H
回転多角形	44H
三角形	45H
矩形	46H
円	47H
円弧	48H
扇形	49H
楕円	4AH
楕円弧	4BH
楕扇形	4CH
ペイント1	4DH
ペイント2	4EH
ポイント識別	4FH
弓形1	50H
弓形2	51H

▼表II-2-4　フォントオペレーション

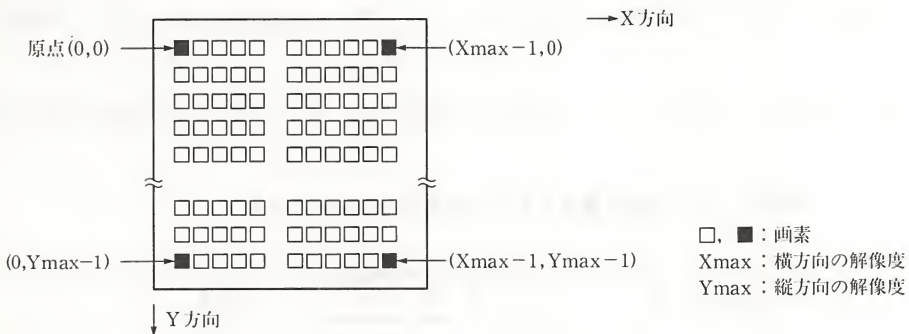
機能名称	機能コード
文字列	60H
追加文字列	61H
文字列1	62H
追加文字列1	63H
文字列2	64H
追加文字列2	65H
任意文字表示	66H

2.2　グラフィック BIOS の基本機能と用語

ここでは，EGB の基本的な機能や用語について解説します．なお，以下の解説では，グラフィック画面を画面と略記します．

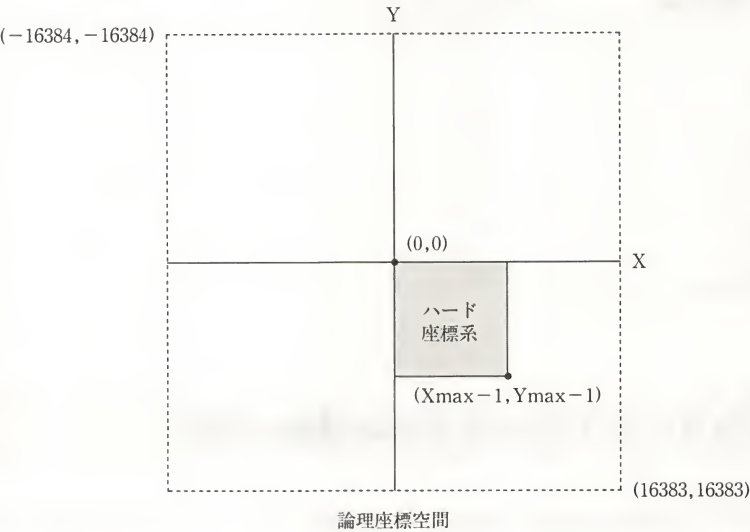
● EGB のハード座標系と論理座標空間

EGB の各画素の位置は，画面上の左上端の画素（ピクセル＝画面上に表示している色の最小範囲の領域）を原点とし，X 軸方向が右向き，Y 軸方向が下向きの座標系で表します．この座標系をハードウェア座標系（ハード座標系）といいます．図II-2-1にハード座標系を示します．

▼図II-2-1　ハード座標系

ハード座標系は，基本的にハードウェアの仮想画面の各ページに対応しています．したがって，解像度はハードウェアの画面モードの設定により異なります．

なお，EGB では，画面外の領域（仮想空間）をサポートしています．扱える空間座標は，符号付15ビット整数（内部16ビット）で，この空間を論理座標空間といいます．EGB では座標の指定にこの論理座標空間を使用します．論理座標空間は座標指定は可能ですが，ハードウェアで実際にサポートされている空間を除き描画は行われません．図II-2-2に論理座標空間を示します．

▼図II-2-2　論理座標空間

論理座標空間

●色識別番号とビット数

EGBは，画素の色を指定して図形を描画します．この画素の色を示すデータを，色識別番号と呼びます．色識別番号はダブルワード(32ビット)で表現します．ただし，実際に，表示可能な色数は，ハードウェアの画面モードの設定に依存するので，同時表示可能色数によって，色識別番号に指定できる値の範囲は異なります．

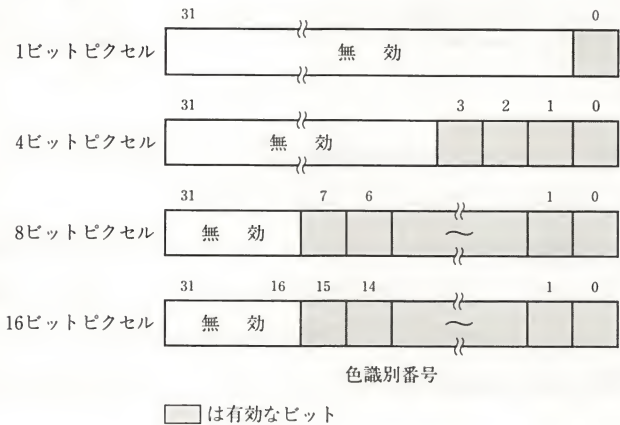
図II-2-3に色識別番号と1ピクセル当たりのビット数の関係を示します．

この図のように，ハードウェアに存在するビット数ぶんだけ下位のビットから有効になります．

なお，2色モード(1ビットピクセル)は，画面をユーザーメモリ上に作成する場合に使用します．

▼図II-2-3　色識別番号と1ピクセル当たりのビット数

色識別番号

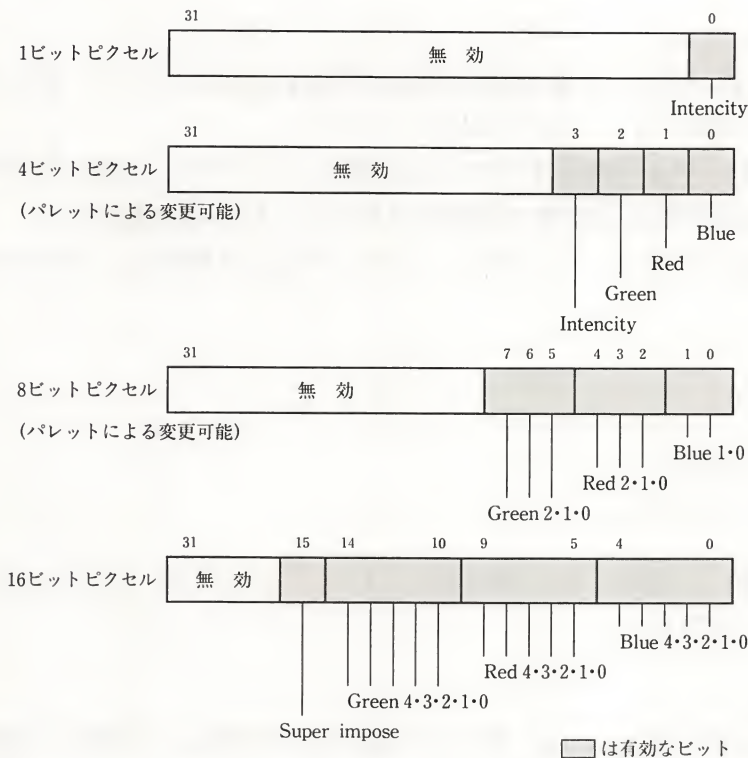
は有効なビット

●パレットレジスタ

EGBでは，パレット機能をサポートしています．同時表示色数が，256色と16色の場合にパレット機能が使用できます．表II-2-5に同時表示色数と色識別番号，パレット機能の関係を示します．

パレット機能が使用可能なモードでは，色識別番号は，実際の描画色(白，赤，青など)そのものではなく，ハードウェアのパレットレジスタのレジスタ番号を示します．パレットレジスタには，実際の表示色が登録されており，画素の表示の際に色識別番号(すなわち，レジスタ番号)を指定すると対応するレジスタに設定されている色(値)がディスプレイに表示されます．

EGBでは，パレットレジスタに格納する色をGRB各8ビット(256階調)で指定できます．

また，パレットレジスタの設定が行われていないデフォルト状態の，各ピクセルのビットの意味を図II-2-4に示します．

▼図II-2-4　ピクセルデータのビットの意味

▼表II-2-5　同時表示色数と色識別番号

同時表示色数	色識別番号の範囲	パレット機能の有無
32768色(16ビットピクセル)	0～15ビット	×
256色(8ビットピクセル)	0～7ビット	○
16色(4ビットピクセル)	0～3ビット	○
2色(1ビットピクセル)	1ビット	×

●表示ページと書き込みページ

ハードウェアでは，VRAMを複数のページに分けて使用することができますが，EGBでもこれに対応して，複数ページの読み書きが可能です．

ページ数が複数ある画面モードの場合，表示ページと書き込みページを別々に設定できます．この場合，書き込みページの指定は一度に1ページのみですが，表示ページは2ページまで設定できます(2枚のページを重ねて表示する)．このときは，2つのページのうちどちらを手前に表示するか(プライオリティ)を設定できます．

●前景色と背景色

EGBでは，一般に，図形などを描く場合に使用される色を前景色といいます．また，何も描画を行わない状態で表示されている色を背景色といいます．

例外として，描画モードがPRESETの場合と，描画時にドットパターンを指定するオペレーションで，ビットが0の部分は背景色で描画されます(ビット1の部分は前景色)．

前景色と背景色は，ベタ塗り，タイル塗り，ハッチング塗りの3種があり，次の4つの組み合せが可能です．

・ベタ塗り
・タイル塗り
・ベタ塗り＋ハッチング
・タイル塗り＋ハッチング

ベタ塗りは，描画した領域を1色で一様に塗りつぶすことです．また，タイル塗りは，タイルパターンに従って塗ります．

●閉領域と面塗色

画面に四角の枠を描く場合のように，閉じられた領域ができる場合，その閉じられた部分を閉領域，その外側の部分を開領域といいます．閉じた領域を塗る場合，その色を面塗色といいます．この場合には，境界(枠)は前景色となります．

●透過色

描画の際にパターンイメージとして存在しても着色されない色を透過色といいます．透明の色というわけです．以前の色の上に重ね書きしたときは，以前の色は消えます．

●画面枠

EGB の説明で使用されている画面枠とは，実際にディスプレイに表示されている範囲のことです．

●クリップ枠とビューポート，マスク

クリップ枠は，EGB のオペレーションが有効な空間領域のことです．描画の範囲として設定した矩形の領域を特にビューポートといいます．任意の図形のクリップには，マスクを使います．

●描画演算

EGB では，描画の際に，描画色をそのまま描くのではなく，各種の演算を行ってさまざまな効果を出すことができます．これを描画演算といいます．表II-2-6に各演算の意味を示します．描画演算の設定は，「描画モードの設定(機能コード 0AH)」で行います．

▼表II-2-6　描画演算と描画

演算名	意　味
PSET	描画図形の色で描画する．
PRESET	背景色で描画する．
OR	描画図形の色データとこれから描画しようとする画面の色データを論理和した色で描画する．
AND	描画図形の色データとこれから描画しようとする画面の色データを論理積した色で描画する．
XOR	描画図形の色データとこれから描画しようとする画面の色データを排他的論理和した色で描画する．
NOT	描画図形の色データを反転した色で描画する．
MATTE	描画図形の色で描画するが，透過色の部分は，描画されない．
PASTEL	描画図形の色データとこれから描画しようとする画面の色データを演算し，水彩のような効果を出す．
OPAQUE	線分パターン，文字パターンの1のビットを前景色で，0のビットを背景色で描画する．
IMPSET	描画図形の色データのうち，スーパーインポーズビットを1にする．
IMPRESET	描画図形の色データのうち，スーパーインポーズビットを0にする．
IMPNOT	描画図形の色データのうち，スーパーインポーズビットを反転する．
MASKSET	描画図形の色データのうち，画面マスク領域のビットを1にする．
MASKRESET	描画図形の色データのうち，画面マスク領域のビットを0にする．
MASKNOT	描画図形の色データのうち，画面マスク領域のビットを反転する．

2.3 グラフィックBIOSオペレーションの共通事項

ここでは，EGBのオペレーションに共通する事項について解説します．

●エントリ

AHに機能コード，DS:ESIにパラメータ列のアドレス，GS:EDIに作業域のアドレスを指定します．

●リターン

AHに終了コードが返され，0の場合は正常終了，0以外の場合は異常終了を示します．

●初期化

グラフィック機能を使用する際には，始めに「初期化(機能コード00H)」，または「仮想画面の設定(機能コード01H)」と「書き込みページの指定(機能コード05H)」の両方を行わなければなりません．このオペレーションでは，ハードウェアおよび，作業領域の初期化を行います．

●スタックサイズ

EGBでは，複雑な図形を処理するために，通常よりも大きなスタックを必要とする場合があります．スタックサイズの目安は，次のとおりです．

機能コード40H未満： BIOS用としては不要です．

機能コード40H以降： 仮想画面の大きさ(横ドット数×縦ドット数)／8を必要とします．ただし，pset，presetの線・文字(文字飾りなし)描画でmaskが無効の場合は，必要ありません．
以下の場合には，さらに増加します．
　paint(機能コード4DH，4EH)：6KB加算
　polygon(機能コード43H，44H，45H)：5KB加算

●データ領域

EGBで扱えるデータ領域は，セグメント境界をまたぐような使用法はできません．また，セグメント境界も使用することはできません．セグメント境界から4バイト離して設定してください．

2.4 グラフィック BIOS リファレンス

EGB について，個別に詳しく解説します．

グラフィックス	20H
初期化	機能コード00H

エントリ　AH　＝00H

GS : EDI　＝作業領域のアドレス

ECX　＝作業領域のサイズ

リターン　AH　＝00H (正常終了時)

説　明　EGB に関係するハードウェアと，作業領域の内容を初期化します．

仮想画面の全ページを色識別番号 0 で消去します．

初期設定の内容を示します．

パラメータ	設定値
解像度	640 × 480 16 色 2 画面
書き込みページ	ページ 0
表示ページ	ハードウェア依存
前景色	最大識別番号 (白)
背景色	色識別番号 0 (黒)
繰抜色	色識別番号 0 (黒)
混色比率	128 (平均)
パレットレジスタ	16 色初期値
ビューポート	画面枠
文字間の空白	0
文字の方向	右
文字の並び	右方向
文字描画 X 座標	0
文字描画 Y 座標	0
面塗りモード	境界ベタ
面塗り色	色識別番号 0 (黒)
描画モード	PEST
スーパーインポーズ	OFF
インポーズ領域	なし
インポーズ輝度	高輝度
ペンの太さ	1
字体	標準
線分パターン	直線
ハッチングパターン	未登録
タイルパターン	未登録

グラフィックス	20H
仮想画面の設定	機能コード01H

エントリ

1．VRAM を使用する場合

AH　　＝01H
AL　　＝ページ (0, 1)
DX　　＝画面モード（ビット 6 ＝ 1 の場合，CRTC 操作なし）

2．ユーザーメモリを使用する場合

AH　　＝01H
AL　　＝80H～83H
DX　　＝仮想画面の横サイズ(32ピクセル単位)
BX　　＝仮想画面の縦サイズ(1 ピクセル単位)
CX　　＝ピクセル(1， 4， 8，16)
DS : ESI＝RAM 領域アドレス

リターン

AH　　＝00H (正常終了時)

説　明

1．VRAM を使用する場合

仮想画面の画面モードなどを設定します．

画面モードはページ単位に設定できますが，複数ページの組み合わせを伴う場合はハードウェア上の制約を受けます．

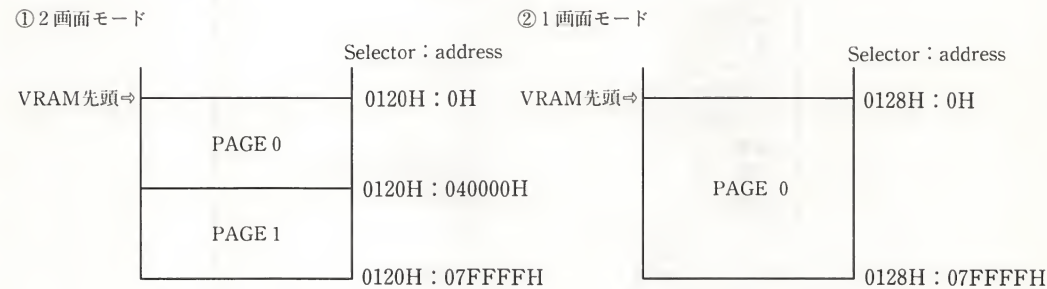
次頁に画面モードと画面の関係を示します．

と VRAM の関係を示します．

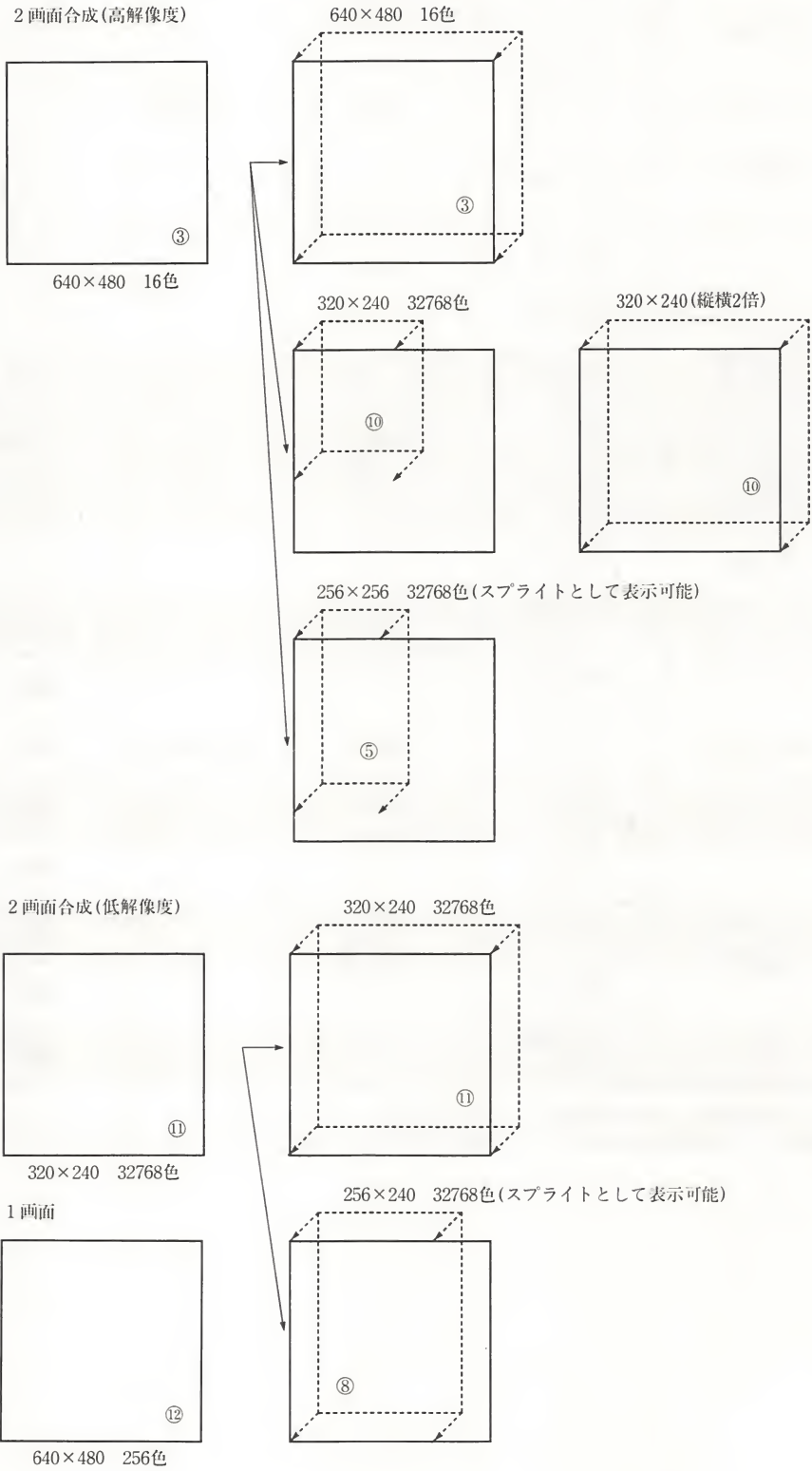
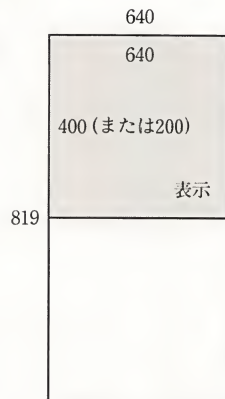
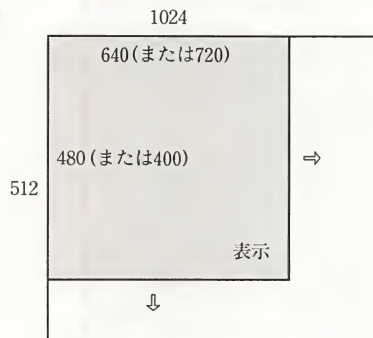
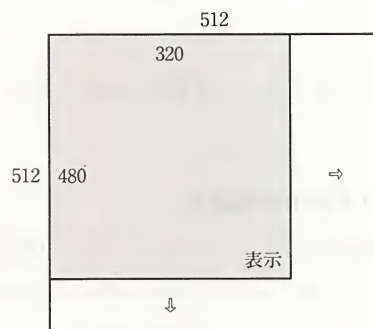
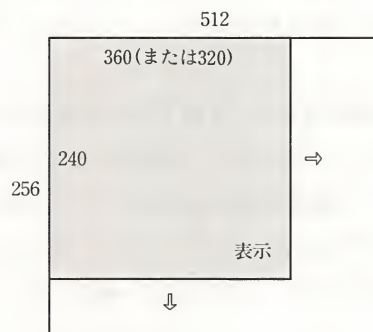
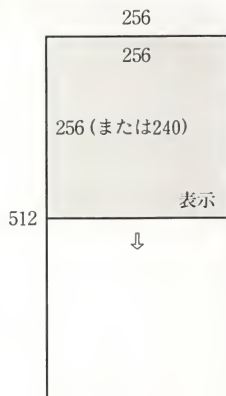
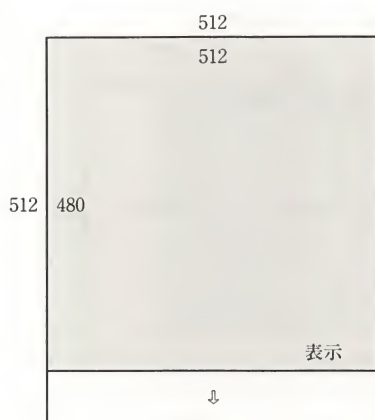
①2画面モード

Selector：address
VRAM先頭⇨
0120H：0H
PAGE 0
0120H：040000H
PAGE 1
0120H：07FFFFH

②1画面モード

Selector：address
VRAM先頭⇨
0128H：0H
PAGE 0
0128H：07FFFFH

画面モード	仮想画面	表示画面	同時表示色	パレット	画面合成
1	640×819	640×400 (ノンインタレース)	16色	16/4096色	1↔1
2		640×200 (ノンインタレース)			2↔2
3	1024×512	640×480 (ノンインタレース)			3↔3，5，10
4		640×400 (ノンインタレース)			4↔4，6
5	256×512	256×256 (ノンインタレース)	32768色	なし	5↔5，3，10
6		256×256 (ノンインタレース)			6↔4，6
7		256×240 (インタレース)			7↔7，9
8		256×240 (インタレース)			8↔8，11
9	512×256	360×240 (インタレース)			9↔7，9
10		320×240 (インタレース)			10↔3，5，10
11		320×240 (ノンインタレース)			11↔8，11
12	1024×512	640×480 (ノンインタレース)	256色	256/1677万色	1画面
13		640×400 (ノンインタレース)			1画面
14		720×480 (インタレース)			1画面
15	512×512	320×480 (ノンインタレース)	32768色	なし	1画面
16		320×480 (インタレース)			1画面
17		512×480 (ノンインタレース)			1画面
18		512×480 (インタレース)			1画面

注）アミかけの部分は，NTSC準拠のビデオ出力が可能．
画面モード1の仮想画面の640×819はEGBがサポートする空間であり，ハードの仕様では640×400に相当．

基本的な画面の組み合せの対応関係を示します.

仮想画面とハードウェア上の制約を図解します.

表示画面の移動なし.
仮想画面の移動なし.

表示画面の移動なし.
仮想画面の移動あり.
(円筒スクロール可能)

表示画面の移動なし.
仮想画面の移動あり.
(円筒スクロール可能)

表示画面の移動あり.
(高解像度のみ)
仮想画面の移動あり.
(球面スクロール可能)

DXのビット6を1にするとCRTCの設定を行いません.

2．ユーザーメモリを使用する場合

仮想画面をVRAMの代わりにユーザーメモリ領域(最大4つの領域が確保できる)に設定することができます．それにはALを80H～83Hにし，画面の大きさを指定します．ユーザーメモリ領域の確保が行われた後は，BIOSの実行時のページ指定で80H～83Hを指定すると，設定したユーザーメモリ領域が使用できます．

DX，BXには画面の横サイズと縦サイズを指定します．CXには，1ピクセルを何ビットで扱うか，DS:ESIには使用するRAM領域のアドレスを指定します．ユーザーメモリ領域に書き込みを行った場合は，画面表示は行われません．描画後に内容をVRAMに転送して始めて表示されます．VRAMに表示する前のバッファ領域などとして使うのが一般的です．

グラフィックス　20H
表示開始位置の設定　機能コード02H

エントリ		
	AH	＝02H
	AL	＝モード(ビット6＝1のときVSYNC待ちなしで設定)
	DX	＝横方向パラメータ
	BX	＝縦方向パラメータ

リターン		
	AH	＝00H(正常終了時)

説　明

表示開始位置などを設定します．

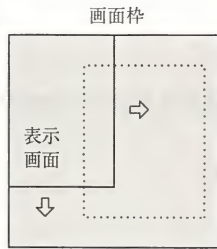
縦／横方向パラメータは，倍率の限界を超えない範囲で，1ドット単位の増減が可能です．

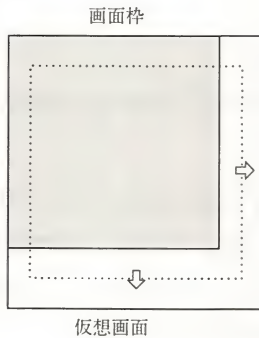
通常はVSYNCを待って設定されますが，モードのビット6が1のときはVSYNCを待たずに設定されます．ただし，その場合アプリケーションがVSYNCを待ってこのオペレーションを実行することが必要で，そうしないと表示回路が動作しなくなることがあります．

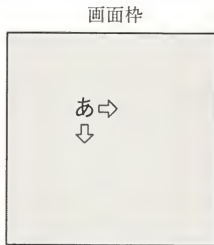
ALの値と設定する内容を示します．

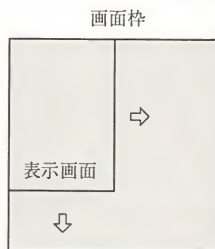
ALの値	設定する内容
0	画面の表示開始位置の設定
1	仮想画面中の移動
2	画面の拡大
3	表示画面の大きさ

各モードの設定の内容を図示します.

画面の表示開始位置の設定

表示画面が画面枠より小さい場合に設定可能.
仮想画面の移動位置が（0，0）に初期化される.

仮想画面中の移動

仮想画面が画面枠より大きい場合に設定可能.
円筒スクロール可能の場合，縦方向に仮想画面を越えた設定が可能.
球面スクロール可能の場合，縦横方向に仮想画面を越えた設定が可能.

画面の拡大

すべての画面モードで設定可能.
最大倍率は16倍.
初期倍率より小さい値を設定することはできない.
仮想画面の移動位置が（0，0）に初期化される.

表示画面の大きさ

表示画面が画面枠より小さい場合に設定可能.
初期画面より小さい値を設定することはできない.
設定パラメータは倍率を考慮したドット単位で指定する.
設定パラメータ値が画面枠を越した場合，画面枠で設定する.

各モードの初期値を示します.

画面モード	仮想画面	表示画面	表示開始位置の設定	仮想画面中の移動	初期倍率 x, y
1	640×819	640×400	×	×	(1, 1)
2		640×200	×	×	(1, 2)
3	1024×512	640×480	×	○	(1, 1)
4		640×400	×	○	(1, 1)
5	256×512	256×256	○	○	(1, 1)
6		256×256	○	○	(1, 1)
7		256×240	○	○	(4, 1)
8		256×240	○	○	(4, 1)
9	512×256	360×240	×	○	(4, 1)
10		320×240	○	○	(1, 1)
11		320×240	×	○	(4, 1)
12	1024×512	640×480	×	○	(1, 1)
13		640×400	×	○	(1, 1)
14		720×480	×	○	(2, 1)
15	512×512	320×480	×	○	(2, 1)
16		320×480	×	○	(4, 1)
17		512×480	×	○	(1, 1)
18		512×480	×	○	(2, 1)

グラフィックス	20H
ビューポートの設定	**機能コード03H**

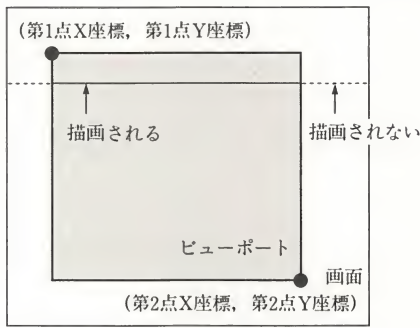
エントリ　AH　＝03H
DS : ESI　＝ビューポートデータのアドレス

リターン　AH　＝00H（正常終了時）

説　明

書き込みページの中で，実際に描画する矩形の範囲を指定します．
論理座標空間内で指定が可能です．
第1点と第2点を対角線とする矩形の内部が描画可能領域となります．
ビューポートデータの形式を示します．

(DS : ESI)

0	SW	第1点X座標
2	SW	第1点Y座標
4	SW	第2点X座標
6	SW	第2点Y座標

直線を描画したときのビューポート内外での扱い

グラフィックス	20H
パレットレジスタの設定	**機能コード04H**

エントリ　AH　＝04H
AL　＝VSYNC フラグ(0：VSYNC 待ちなし，1：VSYNC 待ちあり)
DS：ESI　＝パレットデータのアドレス

リターン　AH　＝00H(正常終了時)

説　明　パレットレジスタに個々の色データを転送します．

設定パレット数以下，その数の組数の色識別番号と Blue/Red/Green の各階調データを並べます．パレットレジスタへのデータ転送を表示中に行うと画面のちらつきが出るため，VSYNC 期間中（輝線の表示を行わない期間）に，転送動作を行うのがいいでしょう．そのためのフラグが AL の VSYNC フラグで，0 のときは VSYNC に関係なくデータ転送を行います．

パレットレジスタはページごとに用意されており，書き込みページに対して有効となります．

そのページが書き込みページであるときに与えられたパレットの内容が，表示ページに指定されたときに表示されます．

パレットデータの形式を示します．

なお，16色2画面表示の場合，画面レイア0側の色識別番号0のパレットは変更できません．また，パラメータを〝VSYNC あり，設定パレット数0〟とすると，VSYNC 待ちのみを行わせることができます．

グラフィックス	20H
書き込みページの指定	機能コード05H

エントリ　AH　＝05H

AL　＝ページ(ユーザーメモリを指定する場合は，80H～83H)

リターン　AH　＝00H(正常終了時)

説明

書き込みページを指定します．

ビューポートの設定を行ったあとで書き込みページの指定をするとビューポートは書き込みページの仮想画面の大きさに再設定されます．したがって，ビューポートの設定は書き込みページの指定の後に設定するのがいいでしょう．

16色パレット使用時には，パレットの有効ページも切り換わります．

表示モード番号指定時に，ALのビット6を1にするとCRTCの設定を行いません．この機能はデバックのときなどに使用されます．

グラフィックス　20H

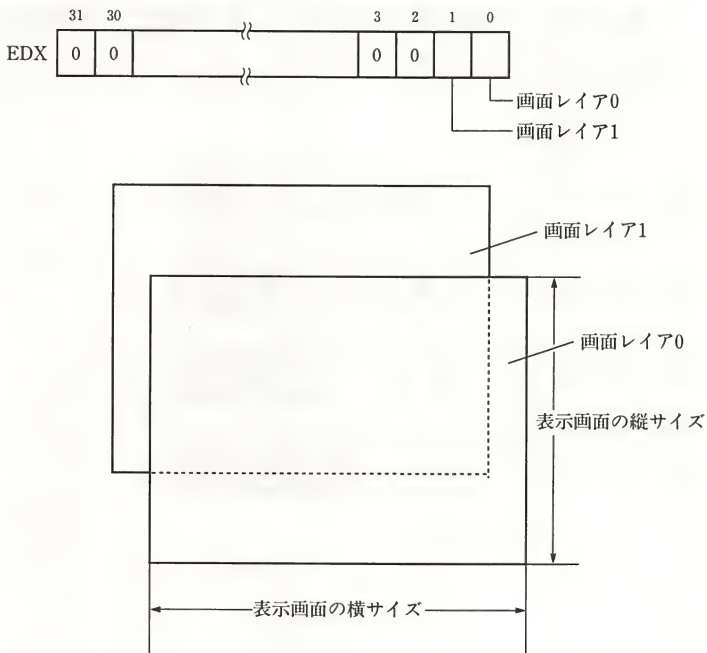
表示ページの指定　機能コード06H

エントリ	AH	＝06H
	AL	＝プライオリティ（0：画面レイア0優先，1：画面レイア1優先）
	EDX	＝表示の有無
リターン	AH	＝00H（正常終了時）

説　明

表示ページの指定を行います．

プライオリティパラメータは，画面レイアの優先度を決めます．ここで，画面レイアと呼んでいるのは，ページのことであり，優先度の高い画面レイアが前面に表示されます．表示の有無は画面レイアごとにEDXの各ビットにより指定できます．ビットが0のときは表示を行わず，ビットが1のとき表示します．

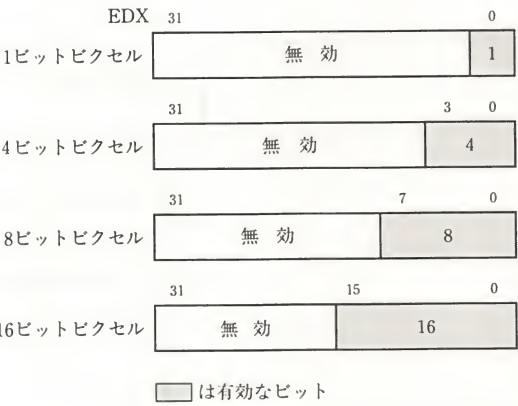
EDXの形式を示します．

グラフィックス	20H
描画色の設定	機能コード07H

エントリ　AH　=07H
AL　=設定色種
EDX　=色識別番号

リターン　AH　=00H（正常終了時）

説　明　前景色，背景色，面塗色，透過色の色識別番号を設定します．
設定色種は，次の4つのうち，いずれかを選んで指定します．

0　：　前景色
1　：　背景色
2　：　面塗色
3　：　透過色

色識別番号の指定は，EDXに通し番号で指定します．1ピクセル当たりのビット数(同時表示色数を意味する)によって，有効ビット数が異なります．
EDXの形式を示します．

グラフィックス	20H
描画色の設定 1	機能コード08H

エントリ　AH　＝08H
AL　＝設定色種
EDX　＝色（IGRB）

リターン　AH　＝00H（正常終了時）

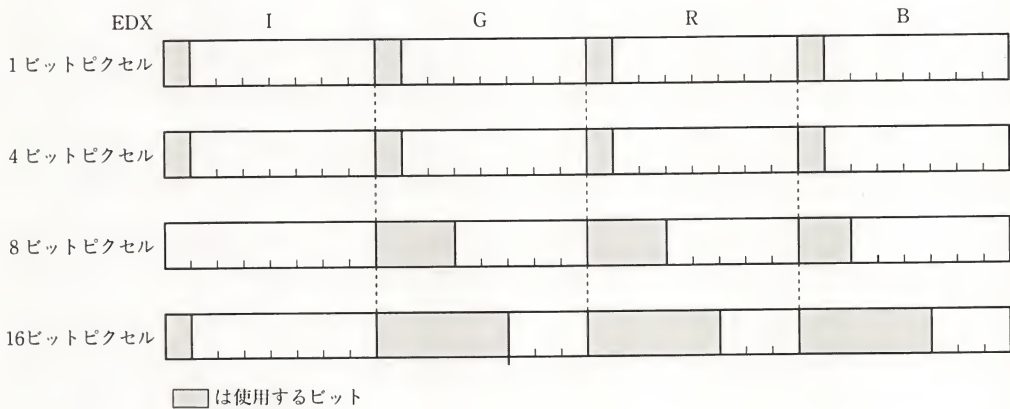
説　明

前景色，背景色，面塗色の色を IGRB で設定します.

なお，256 色，16 色モードでの IGRB の指定は，通し番号に変換され，また，実際に描画される色は，パレットレジスタに設定された色となります.

設定色種は，次の 4 つのうち，いずれかを選んで指定します.

0　：　前景色
1　：　背景色
2　：　面塗色
3　：　透過色

色識別番号の形式を示します.

32 ビットを IGRB 各 8 ビットに分け，それぞれ上位ビットから使用します.
1 ビットピクセルの場合は，I，G，R，B の各最上位ビットがすべて有効です.

グラフィックス　20H

混色比率の設定　機能コード09H

エントリ

AH	＝09H
DX	＝描画色の強調度（0～256）

リターン

AH	＝00H（正常終了時）

説　明

描画色と描画位置色(描画する前にその位置にあった色)との混合比率を設定します．

このオペレーションは，256色と32768色のモードにおいて，「描画モードの設定(機能コード0AH)」で，PASTELモードを選択した場合のみに有効です．水彩画のような透明感のある効果を出すときに用います．描画色の強調度は0が最低値で，この状態で描画しても結果は描画されず，描画位置色のままです．最大値は256で，このときは描画色になります．

また，パレットを使う画面モードの場合は，混色されるのは，色識別番号です．表示色はパレットレジスタの値を参照した色となります．

グラフィックス	20H
描画モードの設定	機能コード0AH

エントリ　AH　＝0AH
　　　　　AL　＝描画モード

リターン　AH　＝00H (正常終了時)

説　明

描画モードを設定します．

ALの値と描画モードの関係を示します．

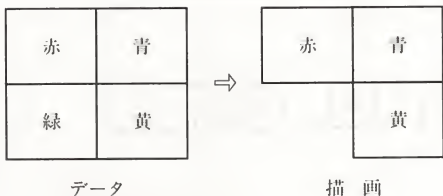
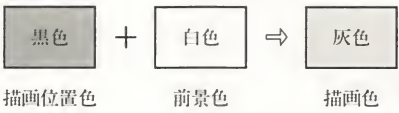
モード番号 (ALの値)	名　称	機　　能
0	PSET	点着色
1	PRESET	点消去(背景色となる)
2	OR	描画位置色とOR
3	AND	描画位置色とAND
4	XOR	描画位置色とXOR
5	NOT	描画色を反転して描画
6	MATTE	透過色を描画しない
7	PASTEL	描画位置色との混合
8	――――	――――
9	OPAQUE	0のビットを背景色，1のビットを前景色で描画
10	IMPSET	スーパーインポーズビットを1
11	IMPRESET	スーパーインポーズビットを0
12	IMPNOT	スーパーインポーズビットを反転
13	MASKSET	画面マスクをセット
14	MASKRESET	画面マスクをリセット
15	MASKNOT	画面マスクを反転

描画モード13～15 (MASK関係) を設定した場合，マスクが有効に設定されていても，実際の描画時にマスクは無効となります．

モード0，2～4，7，9の前景色，モード1，9の背景色，モード6の透過色は，「描画色の設定(機能コード07H)」，「描画色の設定1(機能コード08H)」で設定します．また，モード7の混色比率は「混色比率の設定(機能コード09H)」で設定します．

画面モードのいくつかの例を示します．

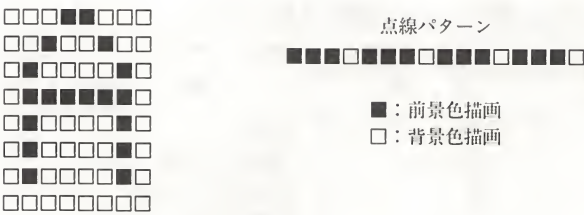
グラフィックデータの書き込みでMATTEモードを指定した場合は次の例で示すとおりになります．

透過色＝緑

PASTEL 描画の例を示します.

文字および線を OPAQUE で描画した場合を示します.

□□□■■□□□
□□■□□■□□
□■□□□□■□
□■■■■■■□
□■□□□□■□
□■□□□□■□
□■□□□□■□
□□□□□□□□

点線パターン

■■■□■■■■□■■■■□■■■■□

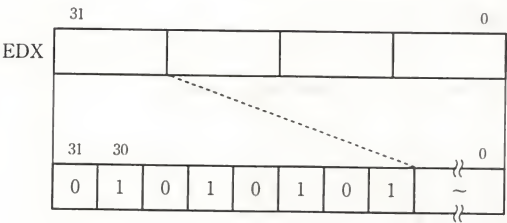
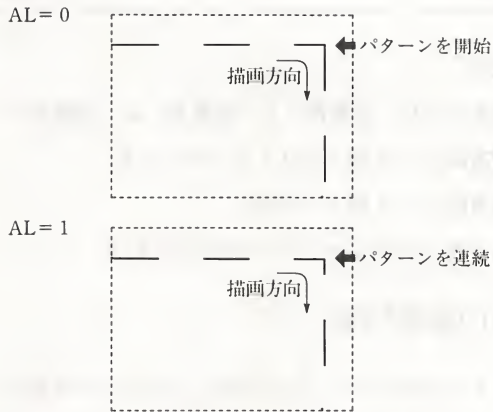
■：前景色描画
□：背景色描画

グラフィックス　20H
線分パターンの設定　機能コード0BH

エントリ	AH	＝0BH
	AL	＝角点処理(0：角点スタート，1：連続)
	EDX	＝線分パターン
リターン	AH	＝00H(正常終了時)

説　明

線分パターンを設定します.

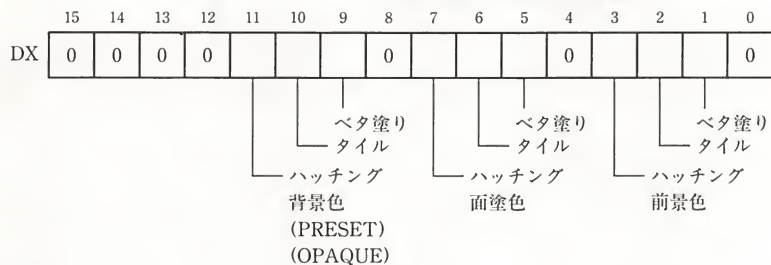
角点処理とは，四角形の角などでパターンを連続して描画するか，一度パターンを切断したうえで，角点から再びパターンを始めるかを指定するものです.

線分パターンの形式を示します.

角点処理の意味を示します.

EDX の各ビットが 0 の場合は，OPAQUE なら背景色で描画します．1 の場合は描画モードに応じて論理演算して描画します.

グラフィックス 20H
面塗りモードの設定 機能コード0CH

エントリ　AH　＝0CH
　　　　　DX　＝面塗りモード

リターン　AH　＝00H（正常終了時）

説　明　面塗色の外，背景色，前景色の塗り方を指定します.
DX の形式を示します.

該当するビットを 1 にして BIOS コールを行います．全ビットが 0 の場合には描画は行われません.

グラフィックス	20H
ハッチングパターンの設定	**機能コード0DH**

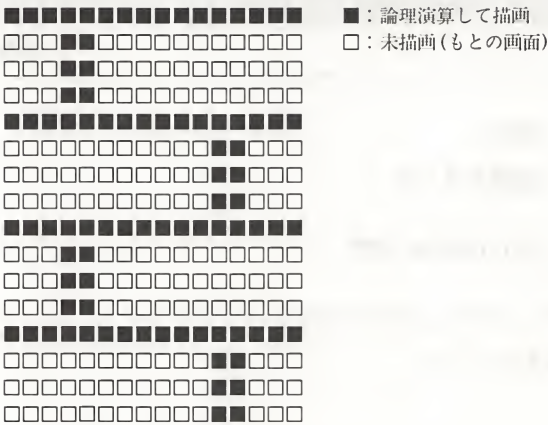
エントリ	AH	＝0DH
	AL	＝設定色（0：前景色，1：背景色，2：面塗色）
	BH	＝横方向ドット数：8n(1<＝n<＝4)
	BL	＝縦方向ドット数：1～32
	DS:ESI	＝ハッチングパターンデータのアドレス
リターン	AH	＝00H(正常終了時)

説　明

ハッチングパターンをメモリ上に作成し，そのデータのアドレスをBIOSに通知します．ハッチングデータは画面の1ピクセルをハッチングパターンの1ビットに対応させて作成します．横方向ドット数は，8ビット単位に設定します．各ビットとも，0で描画しない，1で論理演算して描画する，という指定になります．

ハッチングの例(16×16のデータ)を示します．

■■■■■■■■■■■■■■■■
□□□■■□□□□□□□□□□□
□□□■■□□□□□□□□□□□
□□□■■□□□□□□□□□□□
■■■■■■■■■■■■■■■■
□□□□□□□□□□□■■□□□
□□□□□□□□□□□■■□□□
□□□□□□□□□□□■■□□□
■■■■■■■■■■■■■■■■
□□□■■□□□□□□□□□□□
□□□■■□□□□□□□□□□□
□□□■■□□□□□□□□□□□
■■■■■■■■■■■■■■■■
□□□□□□□□□□□■■□□□
□□□□□□□□□□□■■□□□
□□□□□□□□□□□■■□□□

■：論理演算して描画
□：未描画(もとの画面)

グラフィックス	20H
タイルパターンの設定	機能コード0EH

エントリ

AH　＝0EH
AL　＝設定色（0：前景色，1：背景色，2：面塗色）
BH　＝横方向ドット数：8n(1<＝n<＝4)
BL　＝縦方向ドット数：1～32
DS:ESI　＝タイルパターンデータのアドレス

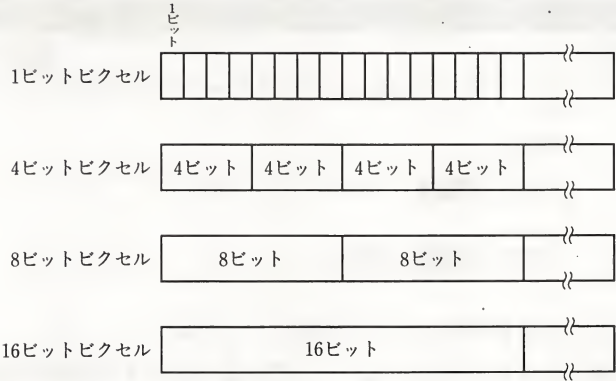
リターン

AH　＝00H(正常終了時)

説明

タイルパターンをメモリ上に作成し，そのデータのアドレスをBIOSに通知します．

タイルパターンデータはVRAMの構成と同じピクセル配置で作成します．画面の1ピクセルをタイルパターンの1ピクセルに対応させて作成します．

タイルパターンのデータの並べ方を示します．

グラフィックス　20H

画面マスク領域の設定　機能コード0FH

エントリ　AH　＝0FH
DS:ESI　＝画面マスク領域のアドレス

リターン　AH　＝00H(正常終了時)

説　明

マスク領域を設定します．

この領域に1ビットピクセルのデータ(2値)を各種のオペレーションで描画することにより，マスクが定義できます．

画面マスク領域の領域サイズは，仮想画面のサイズに応じて，次の式で求められます．

領域サイズ＝((仮想画面横サイズ＋7)×仮想画面縦サイズ)／8

グラフィックス　20H

画面マスクの設定　機能コード10H

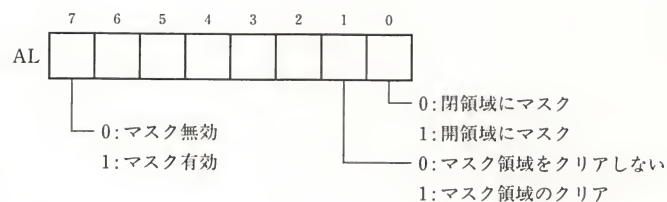
エントリ　AH　＝10H
AL　＝マスク機能

リターン　AH　＝00H(正常終了時)

説　明

画面マスクを有効にしたり，解除したりします．

閉領域をマスクした場合は，マスク領域の値が0の部分は描画の対象となり，1の部分は，描画されません．開領域をマスクした場合は，マスク領域の値が1の部分が描画対象となり，0の部分が描画されません．また，ALレジスタのマスク領域のクリアのビットが1の場合，指定されたマスク領域が0でクリアされます．

ALの形式を示します．

グラフィックス	20H
ペンの設定	機能コード11H

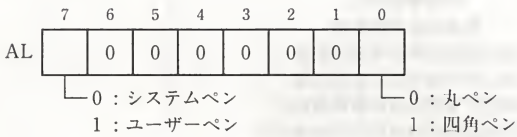
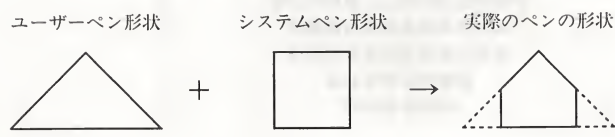
エントリ　AH　＝11H
　　　　　AL　＝ペンモード

リターン　AH　＝00H（正常終了時）

説　明　線描画に使用するペンを設定します．

システムペンとは，EGB のシステムが持っているものです．丸ペンと四角ペンがあります．システムペンの大きさは，ユーザーペンの大きさを限定します．つまり，ユーザーペンとして定義した形の中で，システムペンの大きさに含まれる部分のみが実際のペンの形となります．

ペン形状を変更するためには，ペンの設定をした後，「ペンの太さの設定（機能コード 12H）」を設定する必要があります．

AL の形式を示します．

システムペンとユーザーペンの関係を示します．

グラフィックス　20H
ペンの太さの設定　機能コード12H

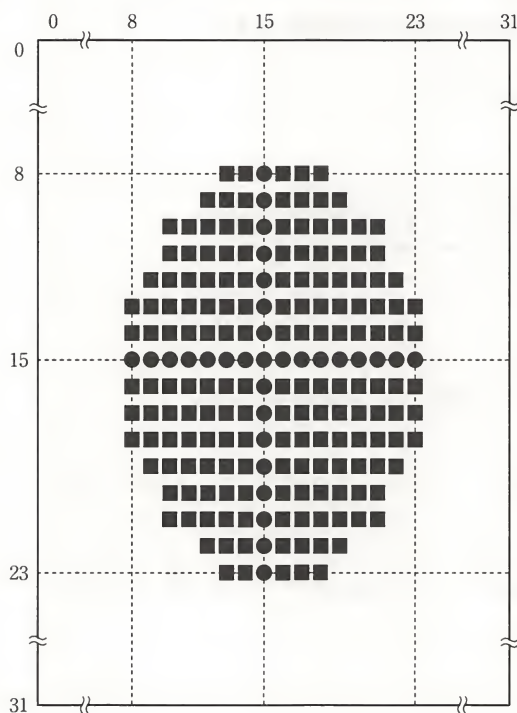
エントリ　AH　＝12H
　　　　　AL　＝ペンの太さ（0～32）

リターン　AH　＝00H（正常終了時）

説　明

システムペンとユーザーペンの太さを設定します．

値が大きいほど，ペン先は太くなります．太さが0の場合，描画は行われません．ペンの中心点は（15，15）ドットの位置にあり，ペンの太さが偶数のときには，右下に太くなります．

ペンの形の例（丸ペンで太さ16の場合）を示します．

グラフィックス	20H
ペンの形状の設定	**機能コード13H**

エントリ　AH　＝13H
　DS : ESI　＝ペンパターンデータのアドレス

リターン　AH　＝00H（正常終了時）

説　明

　ペンの形状を指定します．

　パターンデータは32×32ドットで構成され，ビットが１の部分が形状を表し，演算して描画します．０の部分は，演算しません．

(DS : ESI)

BA	ペンパターンドットデータ (128バイト)

グラフィックス	20H
マスクビットの設定	**機能コード14H**

エントリ　AH　＝14H
　EDX　＝マスクビット

リターン　AH　＝00H（正常終了時）

説　明

　ピクセルの個々のビットについて，描画をマスクします．

　マスクするビットは０，しないビットは１としてEDXに設定します．VRAMのみに有効であり，ユーザーメモリに設定した仮想画面には無効です．マスク領域の設定とは無関係です．

グラフィックス	20H
文字方向の設定	**機能コード15H**

エントリ　AH　＝15H
　AL　＝文字方向

リターン　AH　＝00H（正常終了時）

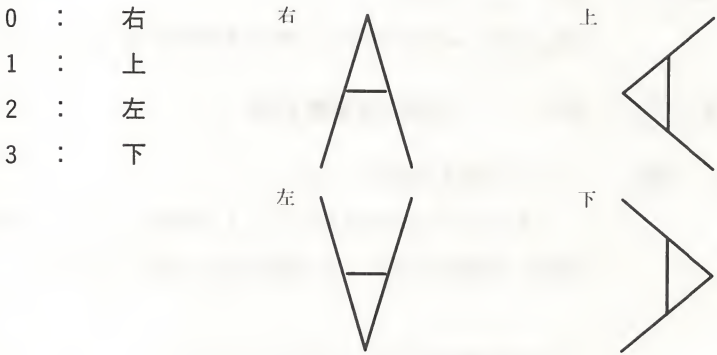
説　明　文字の方向(字体の向き)を設定します.

文字方向は，文字の右側が画面のどの方向を示すかを表します.

ALの値と意味を示します.

0　:　右
1　:　上
2　:　左
3　:　下

右　　上

左　　下

グラフィックス	20H
文字表示方向の設定	機能コード16H

エントリ　AH　＝16H

AL　＝文字表示方向

リターン　AH　＝00H(正常終了時)

説　明　文字の表示方向(並びの向き)を設定します.

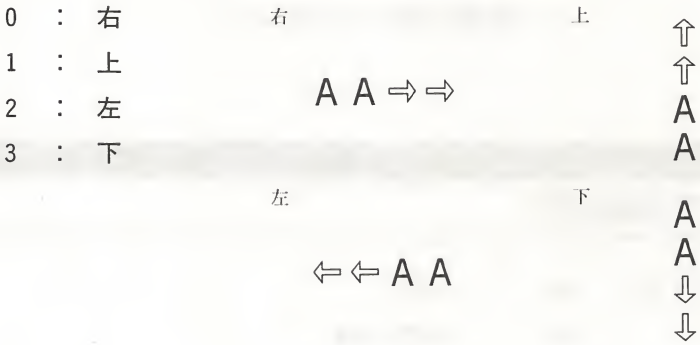
ALの値と意味を示します.

0　:　右
1　:　上
2　:　左
3　:　下

右　AA⇨⇨

上　⇧⇧AA

左　⇦⇦AA

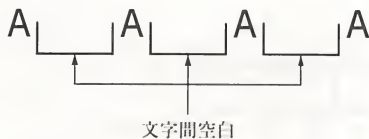
下　AA⇩⇩

グラフィックス　20H

文字間空白の設定　機能コード17H

エントリ	AH	＝17H
	DX	＝文字間空白(ドット数)
リターン	AH	＝00H(正常終了時)

説　明

文字の間の空白の大きさを，任意のドット数で設定します．

ANK と漢字の区別はありません．文字間空白に負の値を設定することにより，重ね書きや後退書きが可能です．

DX は符号付の16ビットで指定します．

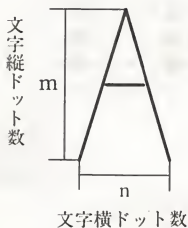
グラフィックス　20H

文字拡大率の設定　機能コード18H

エントリ	AH	＝18H
	DX	＝文字横ドット数(n)
	BX	＝文字縦ドット数(m)
	AL	＝文字種(0：ANK，1：漢字)
リターン	AH	＝00H(正常終了時)

説　明

文字表示の横・縦ドット数を設定します．

ANK と漢字表示について，別々に設定することができます．

グラフィックス 20H
字体の設定 機能コード19H

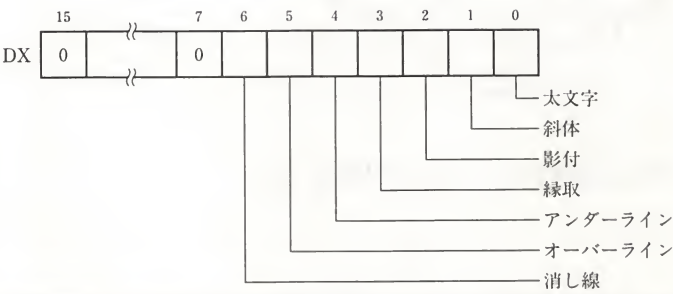
エントリ AH ＝19H
DX ＝字体

リターン AH ＝00H（正常終了時）

説　明

文字の字体を設定します．

DX の該当ビットが1のとき，その字体となります．影付，縁取は背景色で描かれます．

DX の形式を示します．

グラフィックス　20H
スーパーインポーズの設定　機能コード 1AH

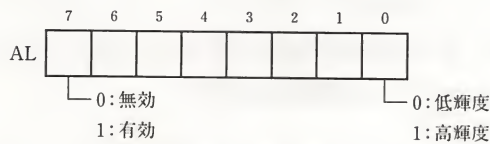
エントリ　AH　＝1AH
AL　＝スーパーインポーズデータ

リターン　AH　＝00H（正常終了時）

説　明

スーパーインポーズ有効／無効，スーパーインポーズ時のビデオ画像の高輝度／低輝度を指定します．

書き込みページで指定したページに対して設定します．輝度の設定は，書き込みページに関係なく設定されます．

AL の形式を示します．

グラフィックス　20H
デジタイズの設定　機能コード 1BH

エントリ　AH　＝1BH
AL　＝ON/OFF（0：OFF，1：ON）

リターン　AH　＝00H（正常終了時）

説　明

画面をデジタイズ状態にするかどうかのスイッチングを行います．

AL に 0 をセットすれば OFF，1 をセットすれば ON になります．書き込みページに対して設定されるので，非表示の画面に画像を取り込むこともできます．

グラフィックス　20H

解像度ハンドルによる仮想画面の設定　機能コード1CH

エントリ　AH　＝1CH
AL　＝CRTC設定（0：する，1：しない）
DX　＝解像度ハンドル

リターン　AH　＝00H（正常終了時）
－1（エラー）

説　明　解像度ハンドルとは，FMTOWNSの高解像度化に伴って追加された機能で，画面の解像度などの情報を取得し細かな画面制御に反映できるようにするためのインターフェースパラメータです．

その値は，システム情報BIOSに参照オペレーションがあり，それによって参照した値を利用して，仮想画面の設定を行うのがこのオペレーションです．実行後，作業領域は次のように初期化されます．

項目	設定内容
書き込みページ	ページ0
表示ページ	ハードウェア依存
前景色	最大色識別番号
背景色	色識別番号0
縁抜色	色識別番号0
混色比率	128（平均）
ビューポート	画面枠
文字間の空白	0
文字の方向	右
文字の並び	右方向
文字描画X座標	0
文字描画Y座標	0
面塗りモード	境界ベタ
面塗り色	色識別番号0
描画モード	PEST
スーパーインポーズ	OFF
インポーズ領域	なし
インポーズ輝度	高輝度
ペンの太さ	1
字体	標準
線分パターン	直線
ハッチングパターン	未登録
タイルパターン	未登録
パレットデータ	16色または256色のとき初期化

グラフィックス　20H

グラフィック描画スタック領域の動的変更　機能コード 1DH

エントリ　AH　＝1DH

AL　＝機能番号（0：要求スタックサイズの取得，1：動的スタックの設定，2：スタック確保/開放イベントの登録）

DS:ESI　＝パラメータ／取得バッファのアドレス

リターン　AH　＝00H（正常終了時）

－1（エラー）

説　明　グラフィック描画スタック領域の動的変更を行うための設定オペレーションです．機能番号と，処理の対応は次のようになっています．

AL＝0（要求スタックサイズの取得）

BIOS が描画に必要なスタックサイズを，次の形式でメモリに取得します．

(DS：ESI)

DW	スタックサイズ

AL＝1（動的スタックの設定）

BIOS が描画時に使用するスタックのサイズやアドレスを，次のパラメータ形式で設定します．このうち，システム動作用スタックサイズは，割り込み処理などで使用するため通常 8,192 バイト程度用意しておけば足ります．

(DS：ESI)

0	DW	スタックサイズ
4	DW	システム動作用スタックサイズ
8	DW	スタック先頭アドレス
12	DW	スタックのセレクタ

AL＝2（スタック確保/開放イベントの登録）

グラフィック BIOS は，描画中にスタックを必要とするとき，登録があればスタック確保イベントを呼び出し，描画終了後スタック開放イベントを呼び出します．

これらはユーザー定義で FAR コールされるプロシージャ（サブルーチン）で，それぞれスタックの取得と開放を行います．これらのサブルーチンでは，レジスタやセレクタの内容を破壊しないようプログラミングする必要があります．

動作の流れは次のとおりです．

① グラフィック BIOS が描画のためにスタックを必要とするとき，スタック確保イベントが呼ばれ，ユーザーには ECX に要求スタックサイズが渡されます．ユーザーは，スタック領域を確保して，「動的スタックの設定（AL=1）」により，グラフィック BIOS に引き渡します．

② グラフィック BIOS は渡されたスタックを使って描画し，終わるとスタック開放イベントを呼び出します．ユーザーはスタックを開放し，リターンします．

これらの登録は次の形式で行われ，登録した内容は,「初期化（機能コード 00H)」,「仮想画面の設定（機能コード 01H)」,「書き込みページの指定（機能コード 05H)」のいずれかのオペレーションが呼ばれたときに解除されます．登録前，および登録解除後には，グラフィック BIOS はイベントを呼び出さず，通常のスタック領域を使用します．

(DS：ESI)

オフセット	型	内容
0	DW	スタック確保イベントのアドレス
4	DW	スタック確保イベントの CS
8	DW	スタック開放イベントのアドレス
12	DW	スタック開放イベントの CS

グラフィックス　20H

デジタイズ画面取り込み位置の補正　機能コード 1EH

エントリ

AH　=1EH
DX　=補正値（X)
BX　=補正値（Y)

リターン

AH　=00H（正常終了時)
　－1（エラー)

説　明

デジタイズ画面取り込み位置を，任意に補正するためのオペレーションです．

補正値は，現在の取り込み位置に対する相対値で，X 方向について右ならば+，左ならば−の値を取ります．Y 方向では，下が+，上が−となります．

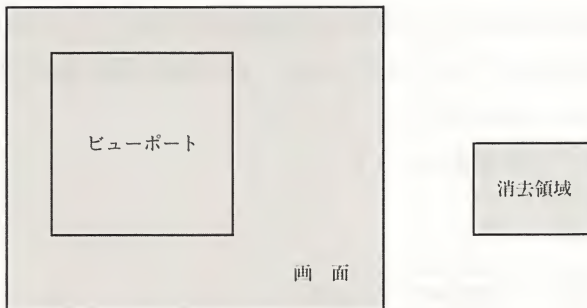
グラフィックス	20H
全画面の消去	機能コード20H

エントリ　AH　＝20H

リターン　AH　＝00H（正常終了時）

説　明　書き込みページの画面全体をクリアし，背景色に塗りつぶします．

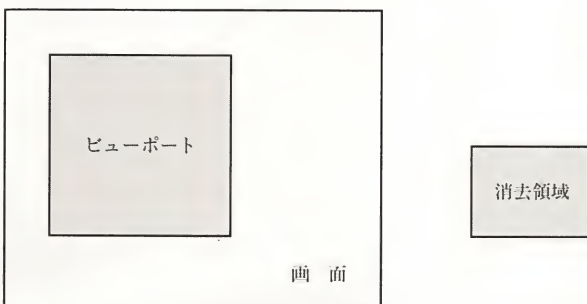
ビューポートの指定を無視して，すべての領域が塗りつぶされます．仮想画面の消去も行われます．マスクの設定は有効です．

2画面合成時に透明以外の背景色を設定すると，優先度の高い方の画面しか表示されないことになります．

グラフィックス	20H
画面の消去	機能コード21H

エントリ　AH　＝21H

リターン　AH　＝00H（正常終了時）

説　明　書き込みページのビューポート内の画面をクリアし，背景色で塗りつぶします．

2画面合成時に，透明以外の背景色を指定すると，優先度の高い方の画面しか表示されなくなります．

グラフィックス	20H
ドットデータの読み出し	**機能コード22H**

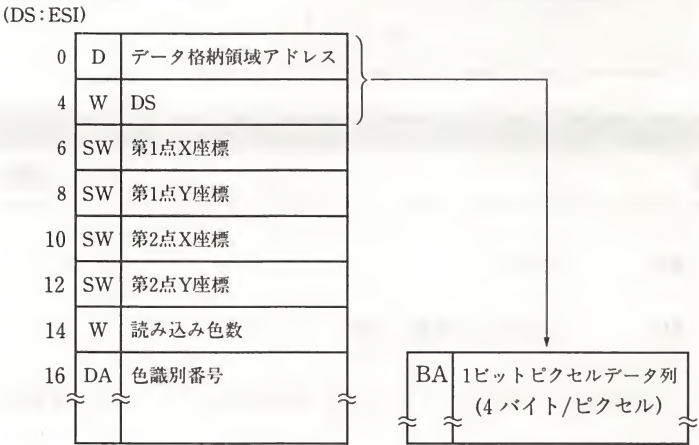
エントリ　AH　＝22H
DS:ESI　＝パラメータのアドレス

リターン　AH　＝00H (正常終了時)

説　明

書き込みページの任意の矩形内のドットデータをモノクロ(2値)で読み出し，テーブルに格納します．

パラメータの設定は，短形の対角線座標と，白とみなす色の色識別番号(32768色モードでは任意の数だけ用意できる)，テーブルのアドレスなどです．データの色識別番号の部分に，白と見なす色識別番号を書くと，そのピクセルは白となり，残りの色はすべて黒となります．色の該当しない部分や仮想画面外のデータは黒として扱われます．

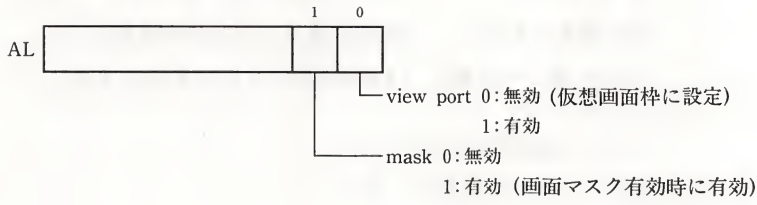
パラメータの形式を示します．

グラフィックス	20H
ドットデータの書き込み	機能コード23H

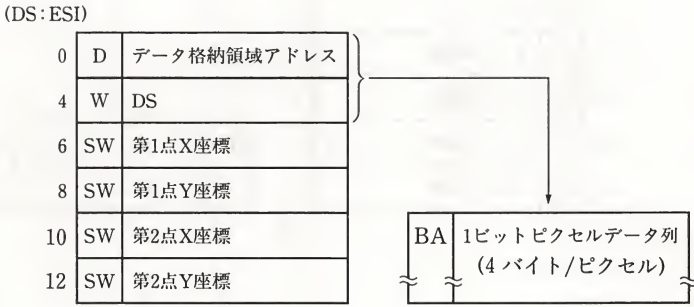
エントリ　AH　　　＝23H
　　　　　AL　　　＝クリップ枠
　　　　　DS : ESI　＝パラメータのアドレス

リターン　AH　　　＝00H（正常終了時）

説　明　書き込みページの任意の位置に，前景色で矩形のドットデータを書き込みます．

「ドットデータの読み出し(機能コード22H)」と組み合わせて使用することにより，図形のコピーができます．ただし，色は単色(前景色で描かれる)になります．

ALによりクリップ枠の有効／無効の設定ができます．

ALの形式を示します．

パラメータの形式を示します．

グラフィックス	20H
ドットデータの読み出し 1	機能コード24H

エントリ　AH　＝24H
DS : ESI　＝パラメータのアドレス

リターン　AH　＝00H (正常終了時)

説　明

書き込みページの任意の矩形内のドットデータを，色識別番号のまま読み出してテーブルに格納します．

読み込み領域は対角線座標で，テーブルのデータ収容領域の大きさは，X と Y の大きさ(ピクセル数)で指定します．画面枠外は，色識別番号 0 として処理されます．

データ格納領域の大きさは，X と Y の大きさによって次の式で求められます．

2 色モード時	(X の大きさ+7)／8×(Y の大きさ)
16 色モード時	(X の大きさ+7)／8×4×(Y の大きさ)
256 色モード時	(X の大きさ)×(Y の大きさ)
32768 色モード時	(X の大きさ)×2×(Y の大きさ)

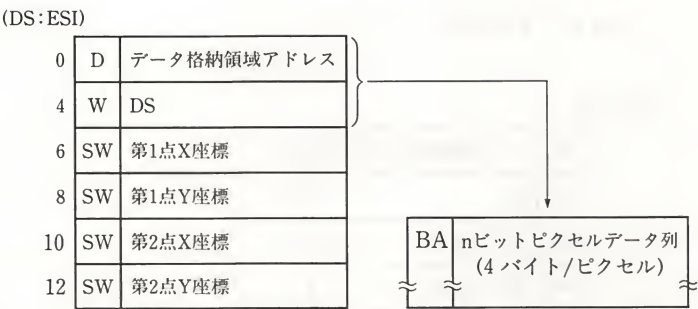
すべて，整数型で計算します．

パラメータの形式を示します．

(DS:ESI)

0	D	データ格納領域アドレス
4	W	DS
6	SW	第1点X座標
8	SW	第1点Y座標
10	SW	第2点X座標
12	SW	第2点Y座標

(0・4 → 指す先)

BA	nビットピクセルデータ列 (4 バイト/ピクセル)

グラフィックス	20H
ドットデータの書き込み 1	機能コード25H

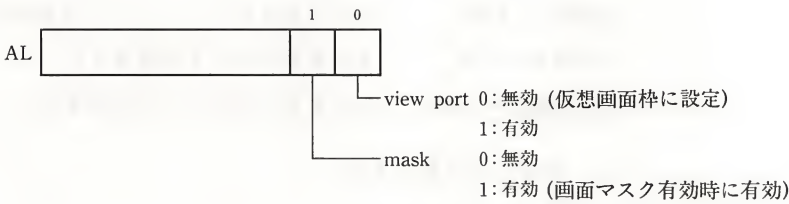
エントリ　AH　＝25H
AL　＝クリップ枠
DS:ESI　＝パラメータのアドレス

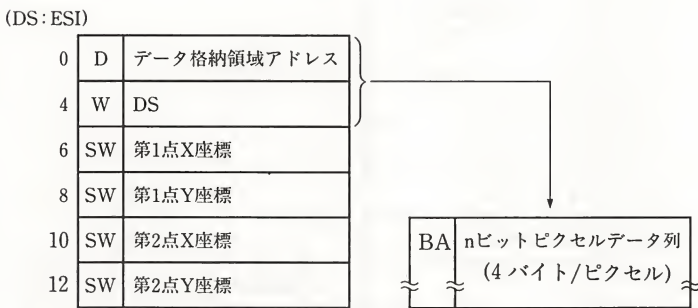
リターン　AH　＝00H（正常終了時）

説　明

「ドットデータの読み出し1（機能コード24H）」に対応する書き込みを行います．

書き込みページの任意の位置に，矩形のドットデータ（色識別番号）を転送します．

AL によりクリップ枠の有効／無効の設定ができます．

AL の形式を示します．

データの形式を示します．

グラフィックス	20H
ドットデータの読み出し 2	機能コード26H

エントリ　AH　＝26H
DS : ESI　＝パラメータのアドレス

リターン　AH　＝00H (正常終了時)

説　明

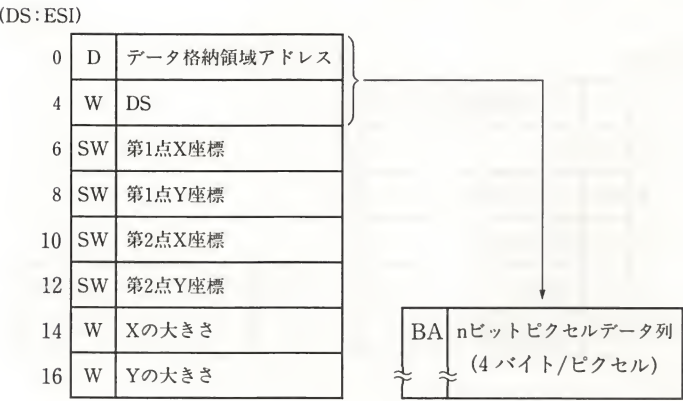
書き込みページの任意の矩形内のドットデータを色識別番号のままで，拡大／縮小して読み込み，テーブルに格納します．

読み込み領域は対角線座標で，テーブルのデータ収容領域の大きさは，XとYの大きさ(ピクセル数)で指定します．画面枠外は，色識別番号0として処理されます．

データ格納領域の大きさは，XとYの大きさによって次の式で求められます．

2色モード時　(Xの大きさ＋7)／8×(Yの大きさ)
16色モード時　(Xの大きさ＋7)／8×4×(Yの大きさ)
256色モード時　(Xの大きさ)×(Yの大きさ)
32768色モード時　(Xの大きさ)×2×(Yの大きさ)

すべて，整数型で計算します．

パラメータの形式を示します．

グラフィックス	20H
ドットデータの書き込み２	機能コード27H

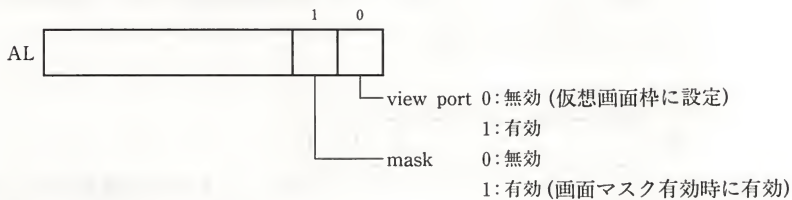
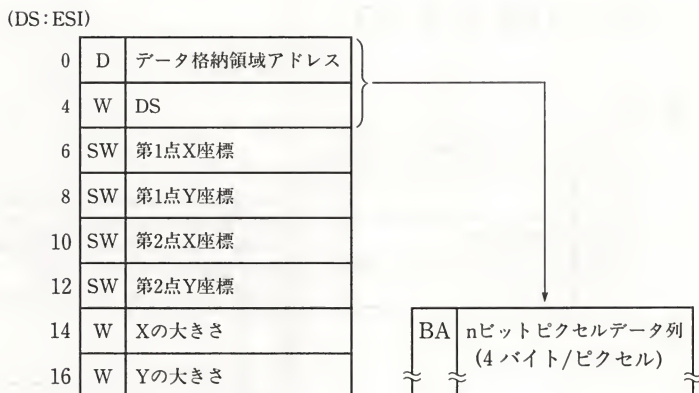
エントリ　AH　　＝27H
AL　　＝クリップ枠
DS：ESI　＝パラメータのアドレス

リターン　AH　　＝00H（正常終了時）

説　明

「ドットデータの読み出し２（機能コード26H）」に対応する，書き込みオペレーションです．

データはテーブルから，書き込みページの任意の矩形の画面領域に転送されます．このとき，X/Y方向のサイズ違いは拡大／縮小によって調整されます．つまり，格納されている大きさと，転送する画面領域の大きさの違いを，拡大／縮小により調整するということです．ALにより，ビューポート，マスクの有効／無効が設定可能です．

ALの形式を示します．

データの形式を示します．

グラフィックス	20H
グラフィックカーソル	**機能コード28H**

エントリ　AH　＝28H
AL　＝クリップ枠
DS : ESI　＝パラメータのアドレス

リターン　AH　＝00H (正常終了時)

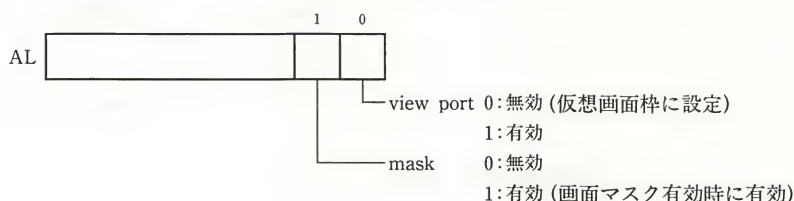
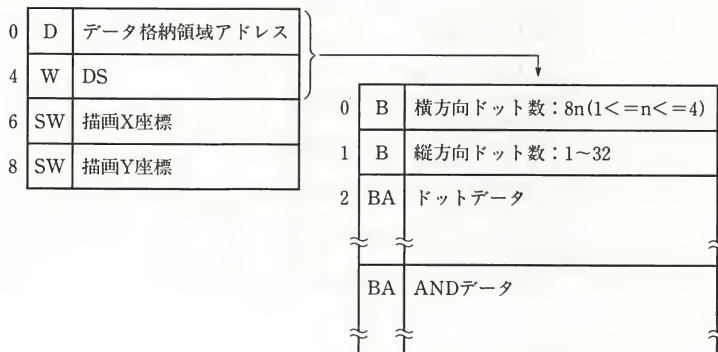
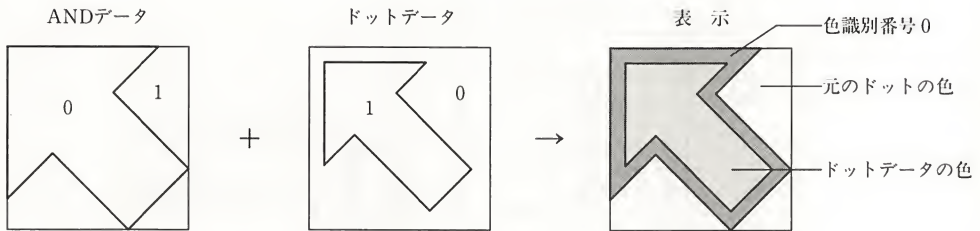
説　明

書き込みページにグラフィックカーソルを書き込みます.

AL により, ビューポート, マスクの有効／無効が設定可能です.

描画座標はカーソルを書き込む画面位置を表し, カーソルの左上端が座標位置に対応します.

データ格納領域には, カーソルパターンを示すドットデータと AND データを配置します. ドットデータと AND データを両方使用することにより, 黒(色識別番号 0)の輪郭を持ったグラフィックカーソルを表示することができます. AND データは, 画面との AND 演算を(ピクセル単位)で行い, AND データが 1 の部分は元のドットを, 0 の部分は色識別番号 0 を書き込みます.

AL の形式を示します.

パラメータの形式を示します.

ANDデータ，ドットデータとグラフィックカーソル表示の関係の例を示します．

グラフィックス	20H
マスクデータの書き込み	機能コード29H

エントリ　AH　=29H
AL　=クリップ枠
DS:ESI　=パラメータのアドレス

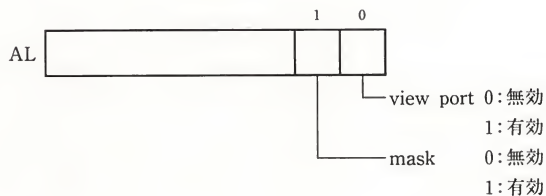
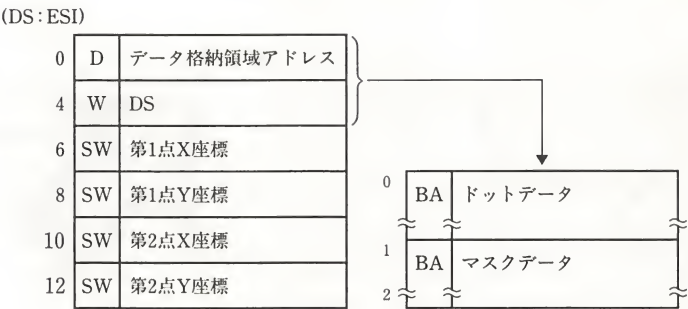
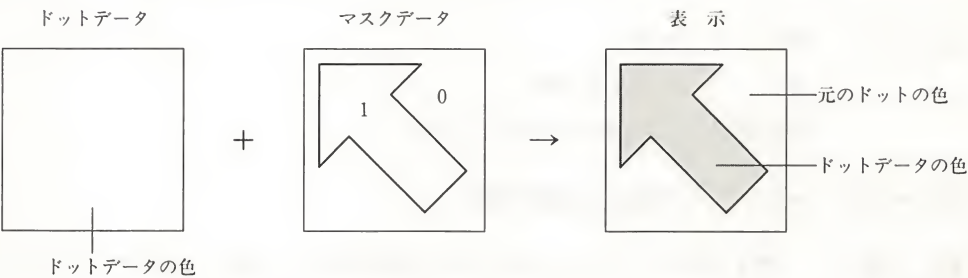
リターン　AH　=00H (正常終了時)

説　明

書き込みページの任意の四角形の画面領域の内部に，各点についてドットデータとマスクデータのANDをとって，書き込みを行います．

マスクデータの値が0の位置は，ドットがマスクされ書き込みは行われません．マスクデータが1の位置のみ，書き込みが許可されます．このオペレーションは，複数の色を使ったカーソルを表示する際に使用します．

マスクデータには，モノクロデータを用意します．

ALの形式を示します．

パラメータの形式を示します.

ドットデータ，マスクデータとグラフィックカーソル表示の関係の例を示します.

グラフィックス	20H
全画面スクロール	機能コード 2AH

エントリ

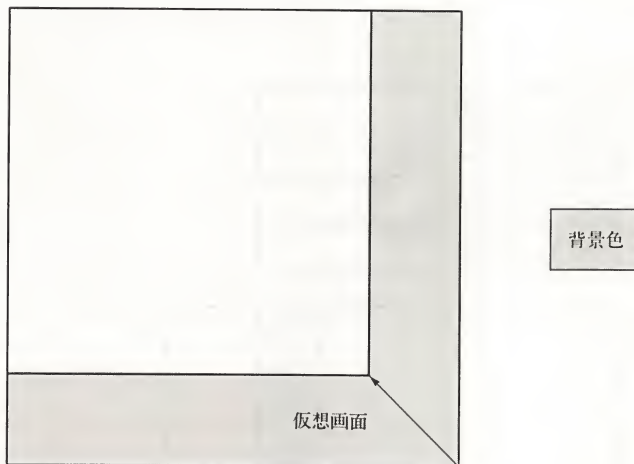
AH　＝2AH
AL　＝背景色描画（0：描画しない，1：描画する）
DX　＝横方向移動ドット数
BX　＝縦方向移動ドット数

リターン

AH　＝00H（正常終了時）

説　明

書き込みページの画面枠を上下左右にスクロールさせます．

ハードウェアによる，円筒スクロールや球面スクロールのような画面の端の連続性はありません．移動により空いた部分は，AL が 1 の場合は背景色となり，0 の場合は元の画面が残ります．

グラフィックス	20H
部分画面スクロール	**機能コード 2BH**

エントリ　AH　＝2BH
AL　＝背景色描画（0：描画しない，1：描画する）
DX　＝横方向移動ドット数
BX　＝縦方向移動ドット数
DS : ESI　＝範囲データのアドレス

リターン　AH　＝00H（正常終了時）

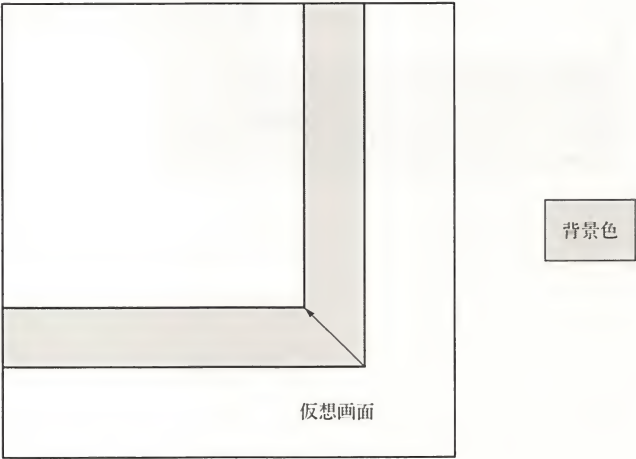
説　明

書き込みページの指定された範囲内を上下左右にスクロールさせます．

移動により空いた部分は，AL が 1 の場合は，背景色となり，0 の場合は元の画面が残ります．ビューポートの外側は影響を受けません．

範囲データの形式を示します．

(DS : ESI)

0	SW	第 1 点X座標
2	SW	第 1 点Y座標
4	SW	第 2 点X座標
6	SW	第 2 点Y座標

グラフィックス　20H

領域の設定　機能コード 2CH

エントリ

AH	=2CH
AL	=内点チェック
DX	=座標点X
BX	=座標点Y
DS:ESI	=多角形データのアドレス

リターン

AH	=00H(正常終了)
AL	=内点チェック結果(0：外点，1：内点)
ECX	=作業領域の大きさ
DX	=X座標点1
BX	=Y座標点1
SI	=X座標点2
DI	=Y座標点2

説　明

画面上の領域に複写，回転，画面ぼかしを行う際には，矩形の領域が対象になります．

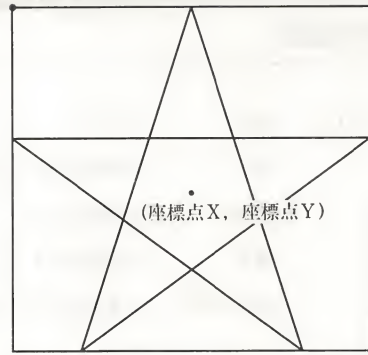
そこで，任意の多角形に対して，上記のような処理を行う場合には，多角形に外接する矩形の大きさ，座標値を調べる必要があります．

このオペレーションは，多角形のデータを与えることにより，対象領域を矩形の対角座標で得るものです．作業領域の大きさを得ることもできます．また，任意の点(DX，BX で指定)に対して，それが対象領域に含まれるかどうかをチェック(内点チェック)することもできます．エントリの AL のビット 7 が 1 の場合内点チェックを行い，結果は AL に返ります．この値が 0 ならば外点，1 ならば内点として判断されたことになります．

得られる作業領域の大きさは，複写などのオペレーションで指定される作業領域のサイズを示すもので，ユーザーが後続のオペレーションを実行する際にこれだけの大きさの作業領域を確保しなければならないことを意味します．実際のプログラミングにおいては，最初から十分な大きさをもつ作業領域を与えておき，EGB によって返されたサイズが，用意した作業領域より大きくないかどうかをチェックする方法が一般的でしょう．

多角形データの形式を示します．

(DS:ESI)

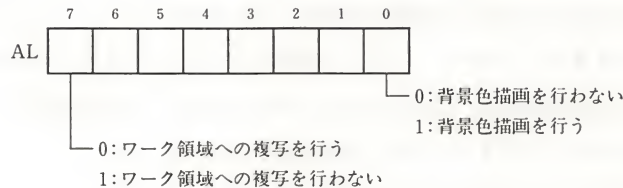
0	W	座標点数(3～256)
2	SW	第1点X座標
4	SW	第1点Y座標
≈	≈	≈

グラフィックス	20H
画面の複写	**機能コード 2DH**

エントリ　AH　=2DH
AL　=機能指定
DS:ESI　=パラメータのアドレス
ES:EBX　=作業領域のアドレス

リターン　AH　=00H (正常終了時)

説　明

「領域の設定(機能コード 2CH)」で設定した領域の中にある図形を，複写データで指定する位置にコピーします．

DS:ESI に指定する座標値は，元の位置との相対位置です．AL によりワーク領域への複写をするかどうか，背景描画(コピー元を背景色でクリアするかどうか)が設定可能です．

AL の形式を示します．

パラメータの形式を示します．

(DS:ESI)

0	B	複写先ページ
1	B	0
2	SW	相対X座標
4	SW	相対Y座標

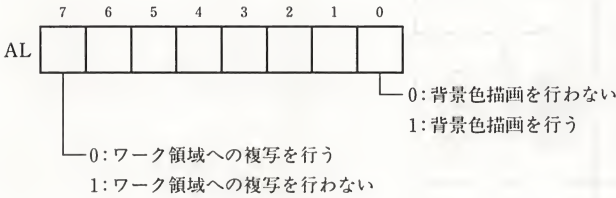
グラフィックス　20H

画面の回転　機能コード 2EH

エントリ　AH　＝2EH
AL　＝機能指定
DS:ESI　＝パラメータのアドレス
ES:EBX　＝作業領域のアドレス

リターン　AH　＝00H(正常終了時)

説　明　「領域の設定(機能コード 2CH)」で設定した領域の中にある図形を，パラメータで指定する位置に回転してコピーします．

ALによりワーク領域への複写をするかどうか，背景描画(コピー元を背景色でクリアするかどうか)が設定可能です．

ALの形式を示します．

パラメータの形式を示します.

(DS:ESI)

0	B	複写先ページ
1	B	0
2	SW	回転中心X座標
4	SW	回転中心Y座標
6	W	回転角度(0～359°)

グラフィックス 20H
画面ぼかし 機能コード 2FH

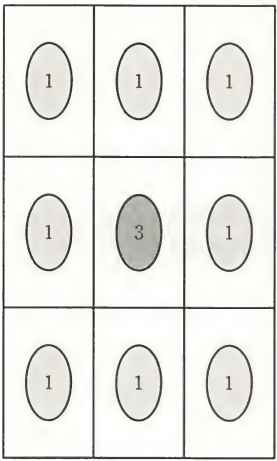
エントリ AH =2FH

ES:EBX =作業領域のアドレス

リターン AH =00H(正常終了時)

説　明 「領域の設定(機能コード 2CH)」で設定した領域の中にある画像データに対し，ぼかし処理をします.

256色と32768色のモードのときに有効です.

ぼかし時の混色比率を示します.

ぼかし時の混色比率は，この図の比率に従って各ピクセルについて 8 方向の値を出し，これを加算して各ピクセルの混色比率を求めます.

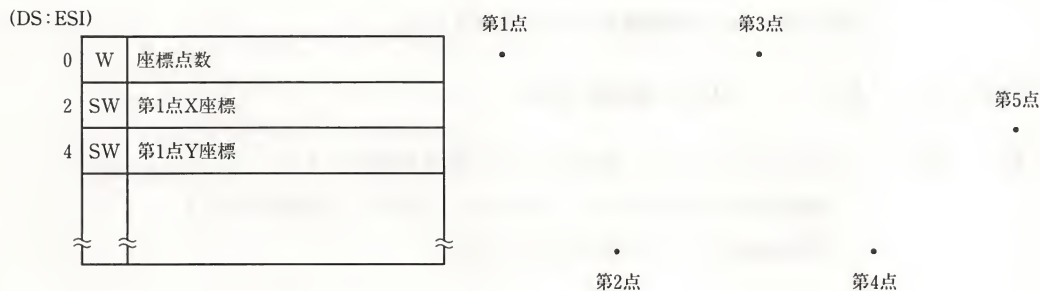
グラフィックス	20H
ポイント	機能コード40H

エントリ　AH　=40H
DS : ESI　=ポイントデータのアドレス

リターン　AH　=00H (正常終了時)

説　明　点を描画します.

「線分パターンの設定(機能コード 0BH)」で指定したラインスタイルの影響は受けません. 座標点数で指定した個数の点が描画されるため, 座標値(X, Y)は描画する点の数だけ指定します.

ポイントデータの形式を示します.

グラフィックス	20H
連続線分	機能コード41H

エントリ　AH　=41H
DS : ESI　=連続線分データのアドレス

リターン　AH　=00H (正常終了時)

説　明　指定された各点を次々に直線で結び, 連続線分を描画します.

連続線分データの形式を示します.

グラフィックス	20H
不連続線分	機能コード42H

エントリ　AH　＝42H
　　　　　DS：ESI　＝不連続線分データのアドレス

リターン　AH　＝00H（正常終了時）

説　明　指定された2点を1組単位として直線を描画します．
奇数個の点が指定されたときには，最後の点は無視されます．
不連続線分データの形式を示します．

グラフィックス	20H
多角形	機能コード43H

エントリ　AH　＝43H
DS：ESI　＝多角形データのアドレス

リターン　AH　＝00H（正常終了時）

説　明

指定された点を次々と結び，さらに最初の点と最後の点を結ぶことにより，多角形を描画します．

座標点数は3点以上必要で，2点以下の場合には描画が行われません．多角形に対して面塗りを行ったときには，多角形描画と面塗りのアルゴリズムの違いにより，境界線と重ならない場合があります．

多角形データの形式を示します．

グラフィックス	20H
回転多角形	機能コード44H

エントリ　AH　＝44H
DS：ESI　＝回転体データのアドレス

リターン　AH　＝00H（正常終了時）

説　明

多角形の回転体を描画します．

座標点は3以上必要で，2以下の場合は描画が行われません．多角形に対して面塗りを行ったときには，多角形描画と面塗りのアルゴリズムの違いにより，境界線と重ならない場合があります．

回転体データの形式を示します．

(DS:ESI)

0	SW	中心点X座標
2	SW	中心点Y座標
4	W	回転角度(0～359°)
6	W	座標点数(3～256)
8	SW	第1点X座標
10	SW	第1点Y座標
		≈

グラフィックス　20H
三角形　機能コード45H

エントリ　AH　＝45H
DS:ESI　＝三角形データのアドレス

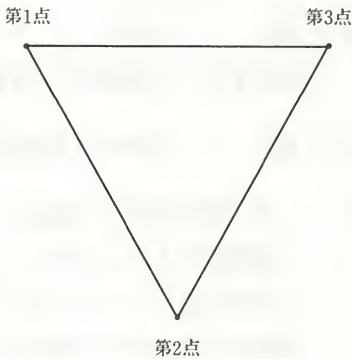
リターン　AH　＝00H(正常終了時)

説　明　指定された3点を頂点とする三角形を描画します．

「多角形(機能コード43H)」を利用しても三角形の描画は可能ですが，このオペレーションでは，頂点数の指定は不要です．三角形に対して面塗りを行ったときには，多角形描画と面塗りのアルゴリズムの違いにより，境界線と重ならない場合があります．

三角形データの形式を示します．

(DS:ESI)

0	SW	第1点X座標
2	SW	第1点Y座標
4	SW	第2点X座標
6	SW	第2点Y座標
8	SW	第3点X座標
10	SW	第3点Y座標

グラフィックス	20H
矩形	機能コード46H

エントリ　AH　=46H
DS:ESI　=矩形データのアドレス

リターン　AH　=00H(正常終了時)

説　明　指定された2点の対角線座標から，矩形を描画します．
「多角形(機能コード43H)」を利用しても矩形の描画は可能ですが，こちらのオペレーションでは，頂点数の指定は不要であり，対角の2点だけの指定ですみます．
矩形データの形式を示します．

(DS:ESI)

0	SW	第1点X座標
2	SW	第1点Y座標
4	SW	第2点X座標
6	SW	第2点Y座標

第1点

第2点

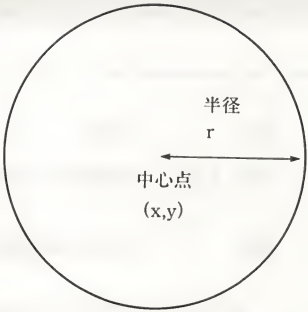
グラフィックス	20H
円	機能コード47H

エントリ　AH　=47H
DS:ESI　=円データのアドレス

リターン　AH　=00H(正常終了時)

説　明　円を描画します．
円データの形式を示します．

(DS:ESI)

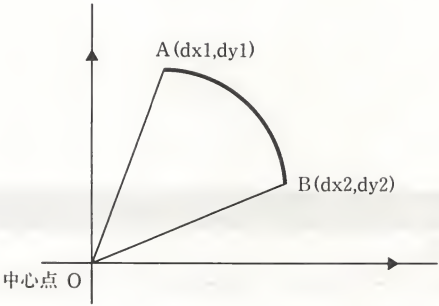
0	SW	中心点X座標
2	SW	中心点Y座標
4	W	半径

グラフィックス 20H
円弧 機能コード48H

エントリ AH =48H
DS:ESI =円弧データのアドレス

リターン AH =00H(正常終了時)

説　明 円のうち開始軸と終了軸で切り取った部分の弧を描きます.
円弧データの形式を示します.

(DS:ESI)

0	SW	中心点X座標
2	SW	中心点Y座標
4	SW	開始軸X成分(dx1)
6	SW	開始軸Y成分(dy1)
8	SW	終了軸X成分(dx2)
10	SW	終了軸Y成分(dy2)
12	W	半径

OA上, OB上にあればdx,dyはどんな値でもよい

グラフィックス	20H
扇形	機能コード49H

エントリ　AH　＝49H
　　　　　DS : ESI　＝扇形データのアドレス

リターン　AH　＝00H (正常終了時)

説　明　円弧の両軸と中心点を結んで扇形を描きます．
　　　　扇形データの形式を示します．

(DS:ESI)

0	SW	中心点X座標
2	SW	中心点Y座標
4	SW	開始軸X成分 (dx1)
6	SW	開始軸Y成分 (dy1)
8	SW	終了軸X成分 (dx2)
10	SW	終了軸Y成分 (dy2)
12	W	半径

グラフィックス	20H
楕円	機能コード 4AH

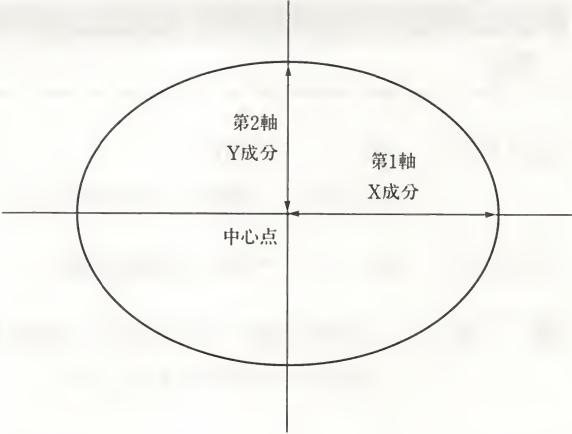
エントリ　AH　＝4AH
　　　　　DS : ESI　＝楕円データのアドレス

リターン　AH　＝00H (正常終了時)

説　明　中心点の座標と，中心点を原点とするX軸座標との交点X，およびY軸座標との交点Yを与えて楕円を描画します．
　楕円データの形式を示します．

(DS:ESI)

0	SW	中心点X座標
2	SW	中心点Y座標
4	SW	第1軸X成分
6	SW	第2軸Y成分

第2軸
Y成分
第1軸
X成分
中心点

グラフィックス　20H
楕円弧　機能コード 4BH

エントリ　AH　　＝4BH
DS:ESI　＝楕円弧データのアドレス

リターン　AH　　＝00H(正常終了時)

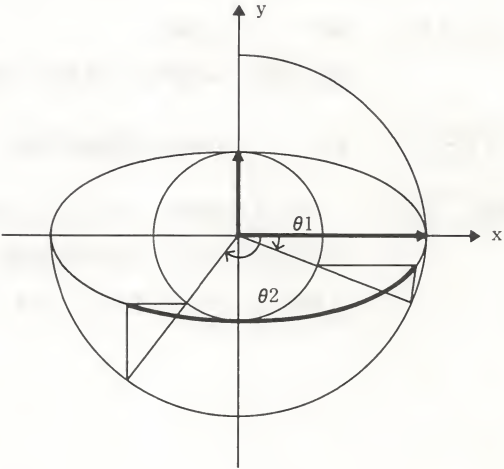
説明

中心点の座標と，中心点を原点とするX軸座標との交点X，およびY軸座標との交点Yを与えて作られる楕円のうち，開始角と終了角で挟まれた弧の部分を描画します．

開始角と終了角の指定は，ともにラジアンであり，ビット31〜ビット16で小数点以上を，ビット15〜ビット0で小数点以下を指定します．

楕円弧データの形式を示します．

(DS:ESI)

0	SW	中心点X座標
2	SW	中心点Y座標
4	SW	第1軸X成分
6	SW	第2軸Y成分
8	FI	開始角(ラジアン)
12	FI	終了角(ラジアン)

開始角($\theta1$)と終了角($\theta2$)は，X 軸正方向から Y 軸正方向にまわる向きの角度をラジアン(32 ビット固定小数点数)で指定します．「円弧(機能コード 48H)」の開始軸，終了軸で指定する方法と異なることに注意してください．次の表に，標準的な角度を固定小数点数で表現した値を示します．

角　度	固定小数点数(2 ワード)
π/6(30°)	0000　86B0H(約0.52360)
π/4(45°)	0000　C910H(約0.78540)
π/3(60°)	0001　0C15H(約1.04720)
π/2(90°)	0001　9220H(約1.57080)
2π/3(120°)	0002　182BH(約2.09440)
π(180°)	0003　243FH(約3.14159)
4π/3(240°)	0004　3055H(約4.18879)
3π/2(270°)	0004　B65FH(約4.71239)
2π(360°)	0006　487FH(約6.28319)

楕円弧の開始点，終了点を求める sin 関数や cos 関数の値は，この表の 2π の値を基準にして計算します．2π を越える角度や負の角度の場合，2π による剰余を計算して，$0\sim2\pi$ の範囲に納まる角度を求めた後，sin 関数や cos 関数の値を求めます．

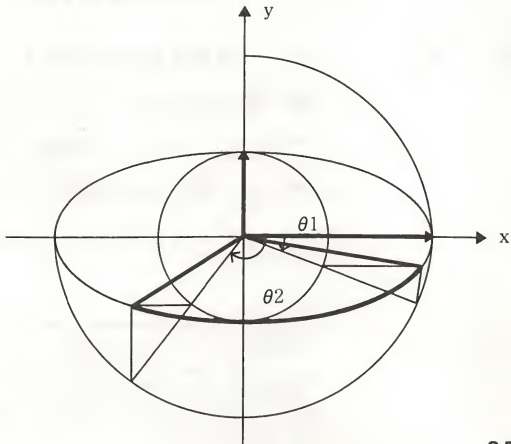
グラフィックス　20H
楕扇形　機能コード 4CH

エントリ　AH　＝4CH
DS : ESI　＝楕扇形データのアドレス

リターン　AH　＝00H (正常終了時)

説　明
楕円弧の両端と中心点を直線で結んで，楕扇形を描画します．
個々の項目の設定は「楕円弧(機能コード 4BH)」と同じです．
楕扇形データの形式を示します．

(DS : ESI)

オフセット	型	内容
0	SW	中心点X座標
2	SW	中心点Y座標
4	SW	第1軸X成分
6	SW	第2軸Y成分
8	FI	開始角(ラジアン)
12	FI	終了角(ラジアン)

グラフィックス 20H
ペイント1　機能コード 4DH

エントリ　AH =4DH
DS:ESI =ペイントデータのアドレス

リターン　AH =00H (正常終了時)

説　明

境界色で囲まれた境界範囲内を塗りつぶします.

境界色数が0のときは，ペイントは行われません．ペイント開始座標は境界範囲内の任意の座標に置くことができます.

ペイントデータの形式を示します.

(DS:ESI)

オフセット	型	内容	
0	SW	ペイント開始X座標	
2	SW	ペイント開始Y座標	
4	W	境界色数	
6	W	0	
8	DA	色識別番号1	境界色数分
12	DA	色識別番号2	
≈	≈		

グラフィックス 20H
ペイント2　機能コード 4EH

エントリ　AH =4EH
DS:ESI =ペイントデータのアドレス

リターン　AH =00H (正常終了時)

説　明

ペイント開始座標点に着色されている色以外の色識別番号を境界色とみなして，境界範囲内を塗りつぶします.

すでに着色されている領域を，別な色で塗りかえるときに使用します．パラメータには，開始点の座標を与えるだけです.

ペイントデータの形式を示します.

(DS:ESI)

オフセット	型	内容
0	SW	ペイント開始X座標
2	SW	ペイント開始Y座標

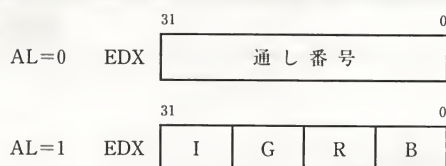
グラフィックス	20H
ポイント識別	機能コード 4FH

エントリ	AH	＝4FH
	AL	＝識別モード(０：通し番号，１：IGRB式)
	DX	＝色識別Ｘ座標
	BX	＝色識別Ｙ座標
リターン	AH	＝00H(正常終了時)
	EDX	＝色識別番号

説　明

座標値で示される点の色識別番号を得ます．

色識別番号は識別モードの指定で，通し番号か，IGRB式のいずれかが選択できます．

ALの形式を示します．

グラフィックス	20H
弓形１	機能コード50H

エントリ	AH	＝50H
	DS：ESI	＝弓形データのアドレス
リターン	AH	＝00H(正常終了時)

説　明

指定された３点を通る円弧を描画します．

第１点と第３点が弧の両端となります．

弓形データの形式を示します．

(DS:ESI)

0	SW	第１点X座標
2	SW	第１点Y座標
4	SW	第２点X座標
6	SW	第２点Y座標
8	SW	第３点X座標
10	SW	第３点Y座標

グラフィックス　20H

弓形２　機能コード51H

エントリ　AH　　＝51H

DS:ESI　＝弓形データのアドレス

リターン　AH　　＝00H（正常終了時）

説　明　第１点，第２点を結ぶ線分を半径とし，第３点を終了軸とする円弧を描きます．

弓形データの形式を示します．

(DS:ESI)

0	SW	第１点X座標
2	SW	第１点Y座標
4	SW	第２点X座標
6	SW	第２点Y座標
8	SW	第３点X座標
10	SW	第３点Y座標

グラフィックス	20H
文字列	**機能コード60H**

エントリ　AH　＝60H

DS : ESI　＝パラメータのアドレス

リターン　AH　＝00H (正常終了時)

説　明

JIS8 ビット文字コード(グラフィックを除く)，またはシフト JIS 漢字コードのいずれか，あるいは両方を含む文字列を表示します．

描画開始位置は，最初の文字の左下の点を示します．

コントロールコードおよび，エスケープシーケンスは，そのコードを表す文字が表示されます．

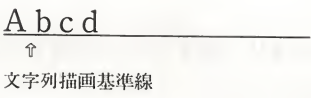
パラメータの形式を示します．

(DS : ESI)

0	SW	描画開始X座標
2	SW	描画開始Y座標
4	W	文字列の長さ
6	BA	文字列データ

Abcd

⇧

文字列描画基準線

グラフィックス　20H
追加文字列　機能コード61H

エントリ　AH　＝61H
　DS：ESI　＝パラメータのアドレス

リターン　AH　＝00H（正常終了時）

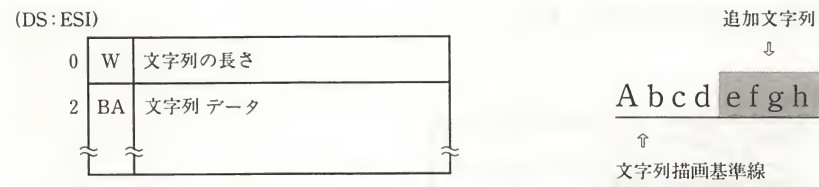
説　明

「文字列（機能コード60H）」のあとに続けて同種の文字列を出力する場合に使用します．

描画開始位置の指定が不要なこと以外は，同等です．

パラメータの形式を示します．

(DS：ESI)

オフセット	型	内容
0	W	文字列の長さ
2	BA	文字列データ

グラフィックス　20H
文字列1　機能コード62H

エントリ　AH　＝62H
　AL　＝ドットフォント（0：8ドット1：16ドット）
　DS：ESI　＝パラメータのアドレス

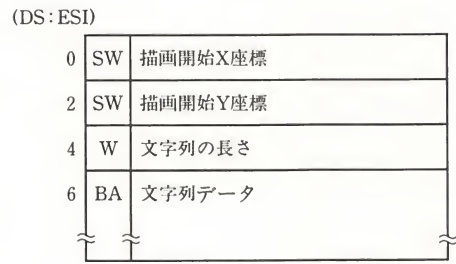
リターン　AH　＝00H（正常終了時）

説　明

グラフィック文字を含むJIS8ビット系文字列を描画します．

パラメータの形式を示します．

(DS：ESI)

オフセット	型	内容
0	SW	描画開始X座標
2	SW	描画開始Y座標
4	W	文字列の長さ
6	BA	文字列データ

グラフィックス	20H
追加文字列1	**機能コード63H**

エントリ　AH　=63H
AL　=ドットフォント(0：8ドット，1：16ドット)
DS:ESI　=パラメータのアドレス

リターン　AH　=00H(正常終了時)

説　明　「文字列1(機能コード62H)」に続けて同種の文字列表示を行う場合に使用します.
開始座標の指定が不要なこと以外は，同等です.
パラメータの形式を示します.

グラフィックス	20H
文字列2	**機能コード64H**

エントリ　AH　=64H
DS:ESI　=パラメータのアドレス

リターン　AH　=00H(正常終了時)

説　明　JIS漢字コードの文字列を表示します.
パラメータの形式を示します.

グラフィックス 20H

追加文字列2 機能コード65H

エントリ	AH	=65H
	DS : ESI	=パラメータのアドレス
リターン	AH	=00H(正常終了時)

説明

「文字列2(機能コード64H)」に続けてJIS漢字コードの文字列を表示する場合に使用します.

開始座標の指定が不要なこと以外は,同等です.

パラメータの形式を示します.

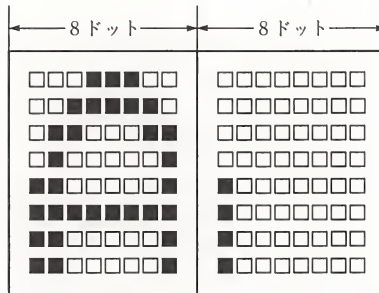
(DS:ESI)

オフセット	型	内容
0	W	文字列の長さ
2	WA	文字列データ

グラフィックス	20H
任意文字表示	機能コード66H

エントリ　AH　＝66H
DX　＝横サイズ(8ドット単位)(8，16，24・・)
BX　＝縦サイズ
DS : ESI　＝文字データのアドレス

リターン　AH　＝00H (正常終了時)

説　明
ユーザーが作成した文字パターンを描画します．
拡大率や字体の設定が行われているときは，その影響を受けます．
文字データの形式を示します．

上図のようなパターンを右のように，16進数表示で登録する．

(DS:ESI)		
0	1C	00
+2	3E	00
+4	63	00
+6	41	00
+8	C1	80
+10	FF	80
+12	C1	80
+14	C1	80
+16		

第 3 章

スプライトBIOS

スプライトは，画面上ではグラフィックデータと合成されて表示されますが，VRAM とは別系統の独立した操作が必要であり，グラフィック BIOS とは別に，スプライト BIOS が用意されています．

この章では，このスプライト BIOS について解説します．

3.1 スプライト BIOS 一覧

スプライト BIOS は，次の 3 種類に分類することができます．

1．初期化オペレーション

スプライトを使用するためのハードウェアの設定を行います．

2．定義オペレーション

スプライトの定義，パレットブロックの設定，位置指定，アトリビュート設定の 4 種類があります．スプライトの定義では，スプライトデータの書き込みを行います．パレットブロックの設定では，パレットカラーデータをセットします．位置指定では，表示する位置を決めます．アトリビュート指定では，スプライトの状態を定義します．

3．表示オペレーション

画面の表示，移動指定，オフセット指定，アトリビュート読み出しの 4 種類があります．このうち画面表示はスプライトの表示を開始します．移動指定は画面上で移動させるときに使用します．移動させるときに，オフセット指定のあるスプライトは，オフセット指定オペレーションが使用できます．

アトリビュート読み出しでは，アトリビュートを参照できます．

表II-3-1に，スプライト BIOS 一覧を示します．

▼表II-3-1　スプライトBIOS一覧

機能名称	機能コード
初期化	00H
画面の表示	01H
スプライトの定義	02H
パレットブロックの設定	03H
位置指定	04H
アトリビュート設定	05H
移動指定	06H
オフセット指定	07H
アトリビュート読み出し	08H

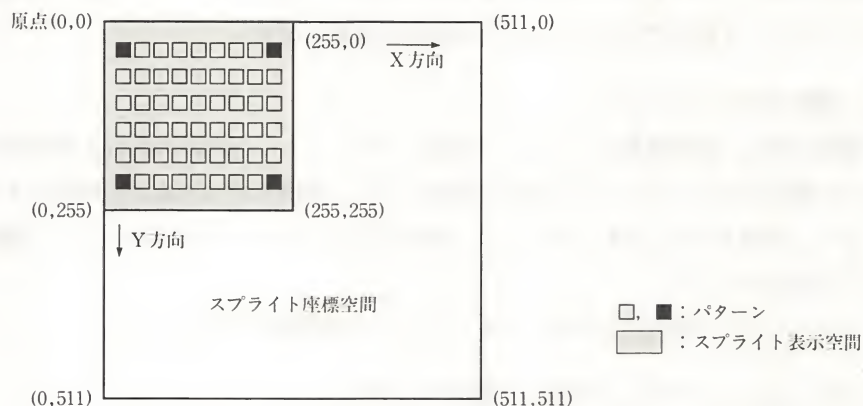
3.2 スプライトBIOSの基本機能と用語

ここでは，スプライトBIOSの基本的な機能や用語について解説します．なお，スプライトのハードウェア仕様については，第I部第4章のスプライトの解説を参照してください．

●スプライト座標空間とスプライト表示空間

スプライトの位置指定は，512×512ピクセルの範囲で可能です．これをスプライト座標空間といいます．ただし，この空間のすべての範囲に表示できるわけではありません．実際の表示は，256×256の範囲に限定されています．これをスプライト表示空間といいます．

図II-3-1にスプライト表示空間とスプライト座標空間を示します．

▼図II-3-1　スプライト表示空間とスプライト座標空間

なお，スプライトの表示には，仮想画面256×512ピクセルの画面モードを使用します．表示可能な画面サイズは，256×256，256×240の２とおりが選べます．

256×240の画面モードでは，実際にスプライトが表示される範囲は，原点(０，０)を起点として(255，239)までの矩形の内部となります．

3.3 スプライトBIOSリファレンス

スプライトBIOSについて個別に詳しく解説します.

スプライト	60H
初期化	機能コード00H

エントリ　AH　=00H

リターン　AH　=00H(正常終了時)

説　明　スプライト関係のハードウェアを初期化し，スプライト画面を色識別番号08000Hで塗りつぶします.

スプライト画面には，256×512ピクセルの仮想画面のページ1を使用しますが，08000Hで塗りつぶすことは，スーパーインポーズビットを立てることを意味します．その結果，スプライト画面は透明になり，背景のグラフィック画面(ページ0)が見えるようになります．スプライトが使用可能な256×512ピクセルの仮想画面の設定は，グラフィックBIOSで行ってください.

スプライト画面の上2ラインは，画面消去用のデータが入ります．初期状態では，前述の08000Hが格納されています．この2ラインにデータを設定すると任意の色でスプライト画面を塗りつぶすことができます．なお，この2ラインにはスプライトを表示することはできません.

スプライト	60H
画面の表示	**機能コード01H**

エントリ
AH =01H
AL =CRT制御
CX =スプライトの個数 (1～1024)

リターン
AH =00H (正常終了時)

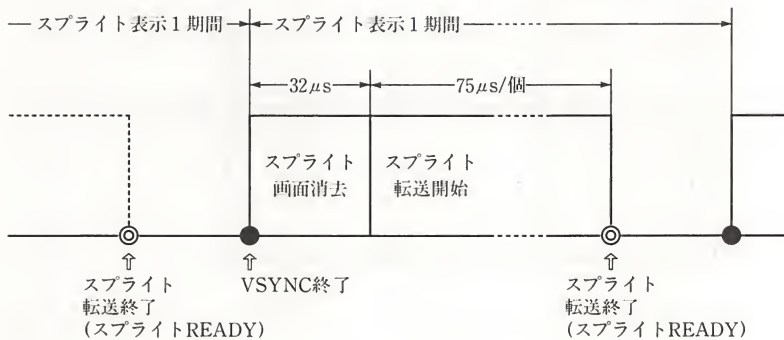
説　明
スプライトは，初期状態では動作しない状態になっています．このオペレーションでは，スプライトの動作の開始，スプライトの個数の設定，スプライト書き込みのタイミング待ちなどの設定を行います．

CXに指定するスプライトの個数は，動作対象とするスプライトの総数を指定するもので，優先度の高いものから指定個数だけ表示します．

スプライトの転送が1表示期間中に終了しない場合，次の表示期間もスプライトの転送をします．

ALの値と意味を示します．

0 : スプライトを動作させない
1 : スプライトを動作させる
2 : スプライトREADY(転送終了)を待つ

スプライトの動作と同期信号の関係を示します．

スプライト	60H
スプライトの定義	**機能コード02H**

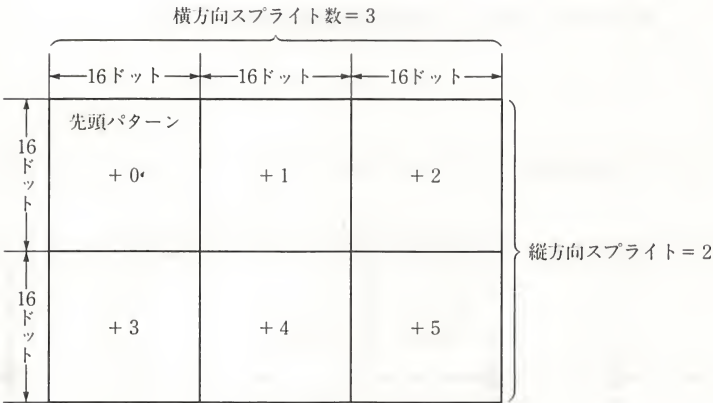
エントリ
AH　=02H
AL　=色数の指定(0：16色，1：32768色)
CX　=先頭パターン番号(128～1024)
DH　=横方向スプライト数
DL　=縦方向スプライト数
DS:ESI　=パターンデータのアドレス

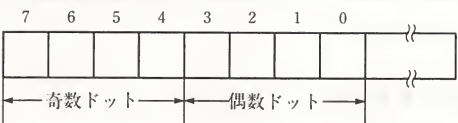
リターン
AH　=00H(正常終了時)

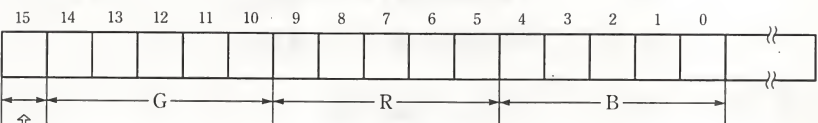
説　明
スプライトのパターンデータを登録します．1つのスプライトが登録できるだけでなく，スプライトを縦横それぞれに複数個並べて，大きなブロックとして登録することができます．横方向スプライト数と縦方向スプライト数には，スプライトを横と縦に並べる数を指定します．

ブロックの先頭位置にあるスプライトを先頭パターン番号で指定すると，連続した番号のスプライトのパターンが登録できます．

32768色ではパターン番号が4の倍数で定義されることに注意してください．

スプライトパターンの登録順の例を示します．

パターンデータの格納順を示します.

32768色中16色時(4ビットピクセル)

4ビットとも 0 がスルー表示.
4ビットで示される16色が, カラーテーブルを経由して32768色に変換されて表示される.

32768色時(16ビットピクセル)

スルービット
0：通常表示
1：スルー表示

スプライト　60H
パレットブロックの設定　機能コード03H

エントリ

AH	＝03H
CX	＝先頭パレットブロック番号(256～511)
DX	＝ブロック数
DS:ESI	＝パレットデータのアドレス

リターン

AH	＝00H(正常終了時)

説　明

指定したパレットブロックへ, カラーデータを転送します.

パレットデータの形式を示します.

(DS:ESI)

スプライト　60H
位置指定　機能コード04H

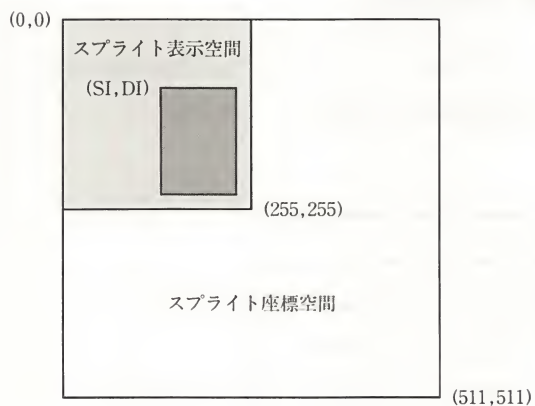
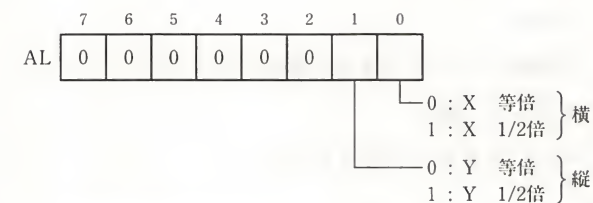
エントリ	AH	＝04H
	AL	＝スプライトサイズ
	CX	＝先頭スプライト番号
	DH	＝横方向スプライト数
	DL	＝縦方向スプライト数
	SI	＝表示横位置(ドット単位)
	DI	＝表示縦位置(ドット単位)
リターン	AH	＝00H(正常終了時)

説　明

スプライト(スプライトブロック)を指定の座標に表示します．

縦横の表示位置は，ブロックの先頭位置にあるスプライトの左上をスプライト座標空間の座標で指定します．

スプライトサイズは，縦横とも等倍と2分の1倍の大きさが選択できます．

ALの形式を示します．

スプライト	60H
アトリビュート設定	**機能コード05H**

エントリ

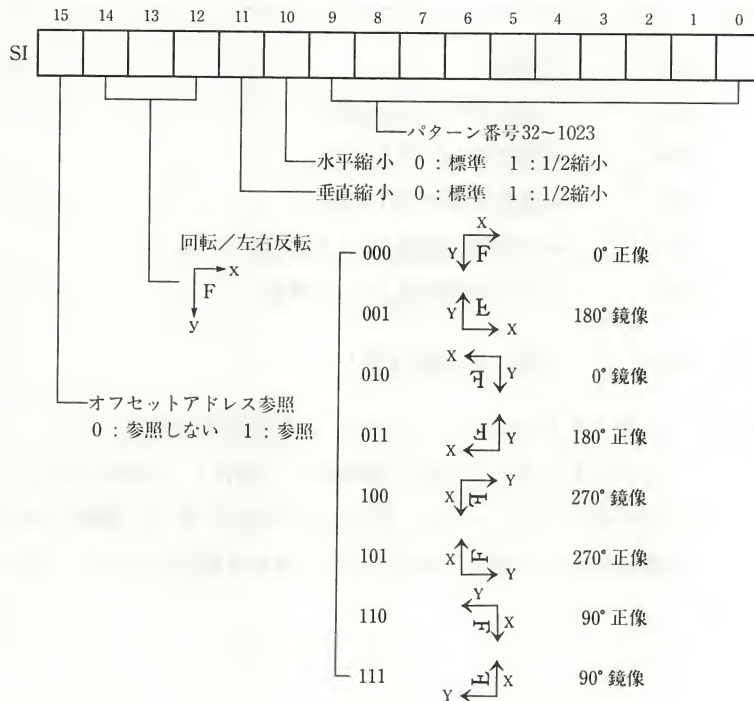
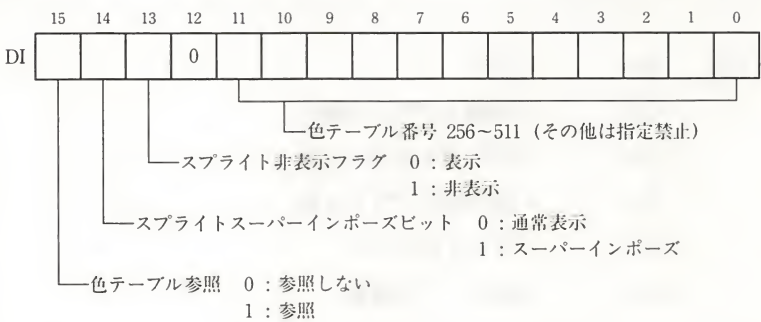
AH =05H
CX =先頭スプライト番号
DH =横方向スプライト数
DL =縦方向スプライト数
SI =アトリビュート
DI =色テーブル番号

リターン

AH =00H(正常終了時)

説　明

スプライト(スプライトブロック)のアトリビュートと色テーブル番号を設定するオペレーションです.

SIの形式を示します.

DIの形式を示します.

色テーブル参照が1のとき16色モード,0のとき32768色モードとなる.

スプライト 60H
移動指定 機能コード06H

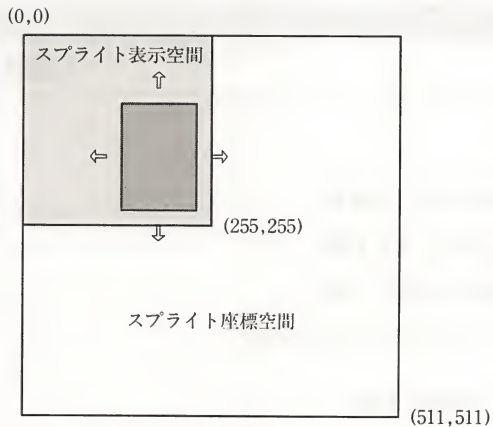
エントリ

AH	＝06H
CX	＝先頭スプライト番号
DH	＝横方向スプライト数
DL	＝縦方向スプライト数
SI	＝X方向移動量(ドット単位)
DI	＝Y方向移動量(ドット単位)

リターン AH ＝00H(正常終了時)

説　明

スプライト(スプライトブロック)を移動させます.

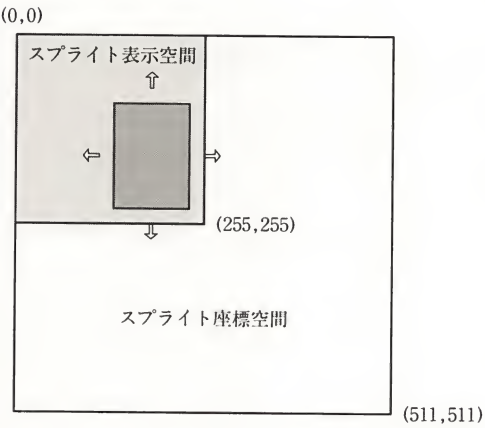
スプライトは「位置指定(機能コード04H)」で移動させることができますが,このオペレーションでは,座標上の相対値を与えて移動させることができます.移動量はドット単位で指定します.負数を指定することもできます.

スプライト　60H
オフセット指定　機能コード07H

エントリ	AH	＝07H
	SI	＝X方向移動量(ドット単位)
	DI	＝Y方向移動量(ドット単位)
リターン	AH	＝00H(正常終了時)

説　明

オフセットアドレス参照を指定してあるスプライトについて，移動量を設定するときに使用します.

移動量は移動指定と同様にドット単位で指定します.

(0,0)
スプライト表示空間
(255,255)
スプライト座標空間
(511,511)

スプライト	60H
アトリビュート読み出し	**機能コード08H**

エントリ

AH	＝08H
CX	＝先頭スプライト番号
DH	＝横方向スプライト数
DL	＝縦方向スプライト数
ES：DSI	＝アトリビュートのアドレス

リターン

AH	＝00H（正常終了時）

説　明

スプライト（スプライトブロック）の先頭座標，アトリビュート，カラーテーブル番号を，アトリビュートのアドレス（転送先先頭）で指定された領域にコピーします．

アトリビュートの形式を示します．

(ES：EDI)

0	W	X座標
2	W	Y座標
4	W	アトリビュート
6	W	カラーテーブル番号
	≈	≈

第 4 章

マウスBIOS

この章では，TOWNS マウスを Towns OS の環境下で使用する場合の，BIOS の機能について解説します．

なお，日本語 MS-DOS の環境下で TOWNS マウスを動作させる場合には，FMR シリーズと共通のマウス BIOS(リアル BIOS)を使用します．この方法については，FMR シリーズの BIOS 解説書を参照してください．

4.1 マウス BIOS 一覧

マウス BIOS は，次の 5 種類に分類することができます．

1．マウスドライバの ON/OFF オペレーション

動作開始，動作終了を行います．

2．カーソル制御オペレーション

水平／垂直移動範囲指定，カーソル形状設定，カーソルの色の設定，タイルパターンの設定，位置の設定などを行います．

3．マウス移動オペレーション

移動距離の読み取り，現在の座標の参照，ボタンの読み取りを行います．

4．ボタン認識オペレーション

押下情報の読み取り，開放情報の読み取り，サブルーチンの登録などを行います．

5．画面オペレーション

パルス数／画素比の設定をします．

表II-4-1にマウス BIOS 一覧を示します．

▼表II-4-1　マウスBIOS一覧

機能名称	機能コード
動作開始	00H
動作終了	01H
表示/消去	02H
位置とボタンの読み取り	03H
位置の設定	04H
ボタンの押下情報の読み取り	05H
ボタンの開放情報の読み取り	06H
水平移動範囲指定	07H
垂直移動範囲指定	08H
形状の設定	09H
移動距離の読み取り	0AH
サブルーチンの登録	0BH
パルス数/画素比の設定	0CH
仮想画面の設定	0DH
書き込みページの設定	0EH
表示色の設定	0FH
タイルパターンの設定	10H
水平消去範囲指定	11H
垂直消去範囲指定	12H
ボタン左右入れ換え状態の設定	13H
加速度検出状態の設定	14H
解像度ハンドルによるマウスの仮想画面設定	15H

4.2 マウスBIOSリファレンス

マウスBIOSについて個別に詳しく解説します．

なお，マウスの動作を正常に行うためには，マウスセンスのルーチンを定期的（約20ms）に呼び出さなければなりません．マウスセンスのルーチンは48Hにあるので，これをFARコールしてください．

また，設定する座標値は，特に断わり書きがない限りハード座標空間です．

マウス	40H
動作開始	機能コード00H

エントリ　AH　＝00H

　　ECX　＝作業領域の大きさ(4096バイト必要)

　　GS : EDI　＝作業領域のアドレス

リターン　AH　＝00H(正常終了時)

説　明　マウスドライバの初期設定を行い，マウスを使用可能な状態にします．

マウス作業領域は，事前に0でクリアしておく必要があります．0クリアを行わないと，正常終了できない場合があります．なお，作業領域には4096バイト必要です．

初期化の設定内容を示します．

パラメータ	設定値
画面モード	0/1とも3に設定
書き込みページ	0
マウスカーソル表示	しない
マウスカーソル位置	画面中心
マウスカーソル形状	システムカーソル
マウスカーソルの色識別番号	最大色識別番号
マウスカーソル中心点	（0，0）
マウスカーソル移動範囲	画面全体
パルス数/画素比	垂直，水平値ともに8
ユーザー定義サブルーチン	未登録

マウス	40H
動作終了	**機能コード01H**

エントリ　AH　＝01H

リターン　AH　＝00H（正常終了時）

説　明

マウスドライバの動作を終了します．

マウスを動かしても反応はなくなり，マウスカーソルも画面上から消えます．

カーソルを表示していない場合は画面に影響はありません．

マウス	40H
表示／消去	**機能コード02H**

エントリ　AH　＝02H

AL　＝表示フラグ（0：消去する，1：表示する，2：マウス表示レベルのデクリメント，3：マウス表示レベルのインクリメント）

リターン　AH　＝00H（正常終了時）

説　明

マウスカーソルの表示／消去を制御します．

マウスカーソルを消去すると，マウスを動かしても画面には表示されませんが，マウスカーソルの位置だけは移動します．マウスカーソルは消えてもマウスの機能は継続しています．

マウス表示レベルは0～65535の値をとります．この値はマウスの表示で1に，消去で0に初期化されます．マウス表示レベルがインクリメントで0から1に変化すると，マウスカーソルが表示され，デクリメントで1から0に変化するとマウスカーソルは消去されます．

マウス	40H
位置とボタンの読み取り	**機能コード03H**

エントリ　AH　＝03H

リターン　AH　＝00H（正常終了時）

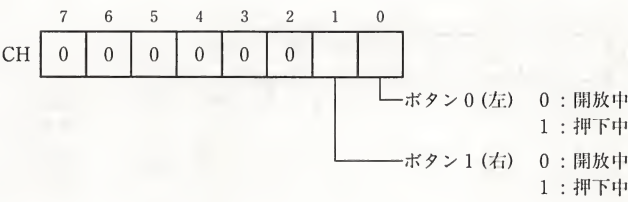
CH　＝ボタンの状態

DX　＝マウスカーソルの水平位置

BX　＝マウスカーソルの垂直位置

説　明　現在のマウスカーソルの位置と，マウスボタンの状態を読み出します．

マウスの状態を連続的に監視する場合に用います．

CH の形式を示します．

マウス	40H
位置の設定	機能コード04H

エントリ　AH　＝04H

DX　＝マウスカーソルの水平位置

BX　＝マウスカーソルの垂直位置

リターン　AH　＝00H(正常終了時)

説　明　マウスカーソルの位置をセットします．

マウスカーソルの表示中に，このオペレーションを実行すると，その位置にカーソルが移動します．「水平移動範囲指定(機能コード07H)」，「垂直移動範囲指定(機能コード08H)」で，水平または垂直の移動範囲が設定されているときはその範囲内の位置に設定されます．

マウス	40H
ボタンの押下情報の読み取り	機能コード05H

エントリ　AH　＝05H

AL　＝状態を読み取るボタン番号(0または1)

リターン　AH　＝00H(正常終了時)

CH　＝ボタンの状態

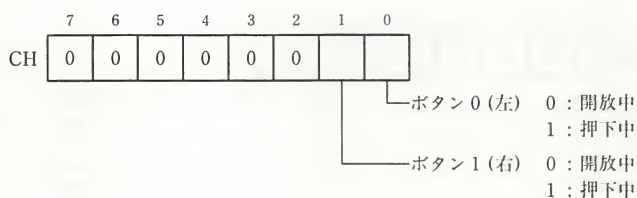
CL　＝ボタンの押下回数

DX　＝X座標

BX　＝Y座標

説 明

左右どちらかのボタンが押下されたときの，各ボタンの状態とマウスカーソルの座標位置を読み取ります．

CH の形式を示します．

CL に返されるボタンの押下回数は，このオペレーションを実行するたびに0にクリアされるので，前回のオペレーションから現在までに押された回数が読みとられます．

座標点(X，Y)はボタンが最後に押された点の座標を返します．

マウス 40H

ボタンの開放情報の読み取り 機能コード06H

エントリ

AH ＝06H

AL ＝状態を読み取るボタン番号(0または1)

リターン

AH ＝00H(正常終了時)

CH ＝ボタンの状態

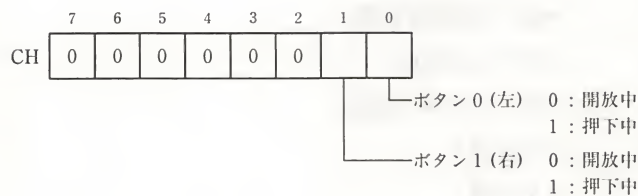
CL ＝ボタンの開放回数

DX ＝X座標

BX ＝Y座標

説 明

左右どちらかのボタンが開放されたときの各ボタンの状態とマウスカーソルの座標位置を読み取ります．

CH の形式を示します．

CLに返されるボタンの開放回数は，このオペレーションを実行するたびに0にクリアされるので，前回のオペレーションから現在までに開放された回数が読みとられます．

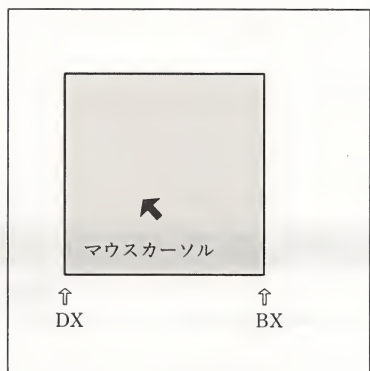
座標点（X，Y）はボタンが最後に開放された点の座標を返します．

マウス	40H
水平移動範囲指定	**機能コード07H**

エントリ	AH	＝07H
	DX	＝最小水平位置
	BX	＝最大水平位置
リターン	AH	＝00H（正常終了時）

説　明

マウスカーソルの水平方向の移動範囲を制限します．

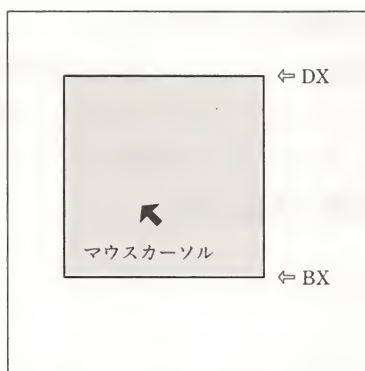
マウスカーソルの移動範囲は最小水平位置から最大水平位置に限定され，はみ出すことができなくなります．なお，このオペレーション実行前にマウスカーソルが範囲外にあった場合，オペレーション実行によってマウスカーソルは前の位置に一番近い境界線の位置に移動します．

仮想画面枠

マウス	40H
垂直移動範囲指定	**機能コード08H**

エントリ
AH　=08H
DX　=最小垂直位置
BX　=最大垂直位置

リターン
AH　=00H(正常終了時)

説明
マウスカーソルの垂直方向の移動範囲を制限します.

マウスカーソルの移動範囲は最小垂直位置から最大垂直位置に限定され，はみ出すことができなくなります．なお，このオペレーション実行前にマウスカーソルが範囲外にあった場合，オペレーション実行によってマウスカーソルは前の位置に一番近い境界線の位置に移動します.

仮想画面枠

マウス	40H
形状の設定	**機能コード09H**

エントリ
AH　=09H
AL　=設定モード(0：システム，1：単色，2：カラー)
DH　=カーソル中心点の水平位置
DL　=カーソル中心点の垂直位置
DS：ESI　=カーソル形状パターンのアドレス

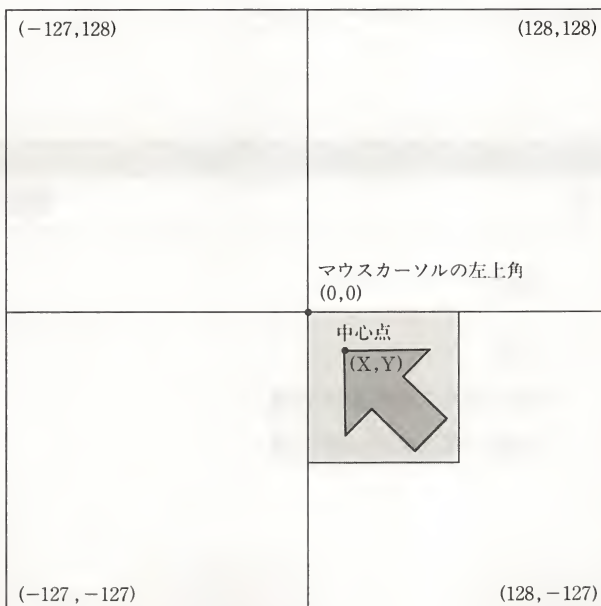
リターン
AH　=00H(正常終了時)

説　明　マウスカーソルのパターンと，中心点(座標の参照位置)を設定します．

ALでは，設定モードを，システム，単色，カラーの3つから選択します．

システムは，次のようなシステム形状に設定します．

単色は単色のカーソルを使用します．カラーは画面モードに応じてピクセル単位に色を付けることができます．なお，システム形状と単色の場合には，「表示色の設定(機能コード0FH)」で，カーソルの色を設定できます．また，単色の場合には，「タイルパターンの設定(機能コード10H)」のオペレーションで，タイルパターンの設定が可能です．

カーソル中心点は，マウスカーソルの座標の参照位置です．マウスカーソルのパターンの左上角(0，0)からの相対位置で任意に設定できます．水平，垂直とも−127〜128(80H〜7FH)の範囲で指定します．

単色，カラーの場合とも，カーソル形状パターンを設定します．

横ドット数は，8，16，24，32のどれか，縦ドット数は1～32が選べます．

単色の場合：ドットデータとANDデータを両方使用することにより，黒の縁どりのあるグラフィックカーソルを表示することができます．グラフィックBIOSの「グラフィックカーソル(機能コード28H)」と同様ですから，詳しくはそちらを参照してください．

パターンの格納に必要なメモリの大きさは，次の式で求められます．

ドットパターンバイト数＝水平ドット数×垂直ドット数／8
ANDパターンバイト数＝水平ドット数×垂直ドット数／8

カラーの場合：複数の色を使うカーソルを表示するのに使用します．詳細はグラフィックBIOSの「マスクデータの書き込み(機能コード29H)」と同様です．

カーソル形状パターンの形式を示します．

(DS:ESI)

0	B	横ドット数／8
1	B	縦ドット数
2	B	マウスカーソル ドットパターン
		マウスカーソル ANDパターン

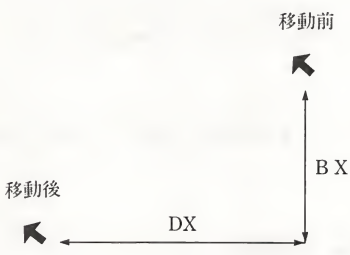
マウス　40H
移動距離の読み取り　機能コード0AH

エントリ　AH　＝0AH

リターン　AH　＝00H
DX　＝水平パルスカウンタの値
BX　＝垂直パルスカウンタの値

説　明

マウスの移動量をカウント値で読み取ります．

水平カウント値，垂直カウント値は，DX，BXによって与えられます．それぞれ，マウスの移動方向によって，正，負どちらかの値となります．DXとBXの値は，このオペレーションの実行の度に0にクリアされるので，前回のオペレーションから現在までの移動量が読みとられます．

マウス　40H
サブルーチンの登録　機能コード0BH

エントリ

AH	＝0BH
DX	＝呼び出し条件
DS:ESI	＝サブルーチンのアドレス

リターン

AH	＝00H（正常終了時）

説　明

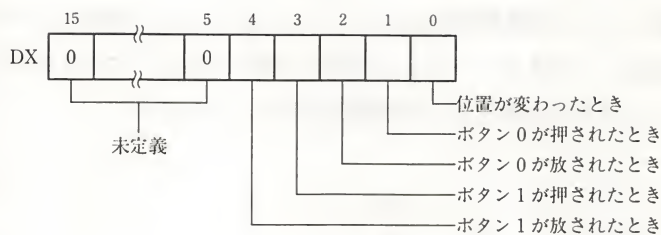
呼び出し条件が成立すると，指定したサブルーチンが起動します．

呼び出し条件のどれかが，1つでも成立するとサブルーチンの呼び出しを行います．

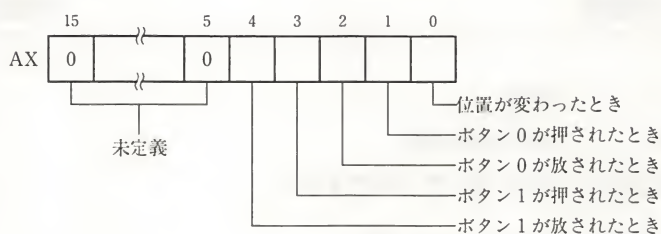
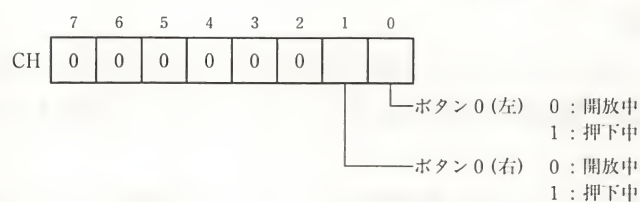
ユーザーサブルーチンは，マウスドライバからセグメント間コール(far CALL)で呼び出されるため，ファーリターンによって制御をもどす必要があります．

このオペレーションを使用すると，マウスの状態をユーザーがプログラムで常時監視する必要がなくなり，条件成立とともに任意のサブルーチンが起動されます．

呼び出し条件は，DXに指定します．各ビットが1の場合に呼び出し条件となります．

DXの形式を示します.

ユーザーサブルーチンがコールされたとき，各レジスタには以下の情報がセットされています.

AX＝呼び出しの原因となった条件
CH＝ボタンの状態
BX＝マウスカーソルの水平位置
DX＝マウスカーソルの垂直位置

AXの形式を示します.

CHの形式を示します.

マウス　40H

パルス数／画素比の設定　機能コード0CH

エントリ	AH	＝0CH
	DH	＝水平値(1～255)
	DL	＝垂直値(1～255)
リターン	AH	＝00H(正常終了時)

説　明

マウスカーソルの移動の程度を，1ドット移動するのに必要なパルス数で設定します．

「動作開始(機能コード00H)」が実行されたとき，マウス移動量の初期値は8パルス／1ドットとなっており，これを変更する場合にこのオペレーションを使用します．パルス数は水平／垂直それぞれ1～255の範囲で指定し，値が小さいほどカーソルの移動の程度が大きくなります．

マウス　40H

仮想画面の設定　機能コード0DH

エントリ	AH	＝0DH
	AL	＝ページ
	DX	＝画面モード番号
リターン	AH	＝00H(正常終了時)

説　明

マウスドライバにページごとに画面モードを通知します．

画面モードはグラフィックBIOSによる仮想画面の設定と同じものを指定します．画面モードとページについては，「第2章　グラフィックBIOS」を参照してください．

このオペレーションにより，次のように初期化されます

マウスカーソル位置	画面中央
マウスカーソル形状	システムカーソル
マウスカーソル移動範囲	画面全体

なお，このオペレーションにより直接画面の表示制御は行われません．表示制御はグラフィックBIOSで行います．

マウス 40H

書き込みページの設定 機能コード0EH

エントリ AH =0EH

AL =書き込みページ

リターン AH =00H(正常終了時)

説　明 マウスカーソルを描画するページをマウスドライバに通知します.

カーソルは書き込みページに描画されるので，グラフィックBIOSで書き込みページを変更した場合にはこのオペレーションを実行する必要があります.

説　明 変更前のページの画面モードと変更後のページの画面モードが異なる場合，以下のように設定します.

マウスカーソル位置	画面中央
マウスカーソル移動範囲	画面全体

マウス 40H

表示色の設定 機能コード0FH

エントリ AH =0FH

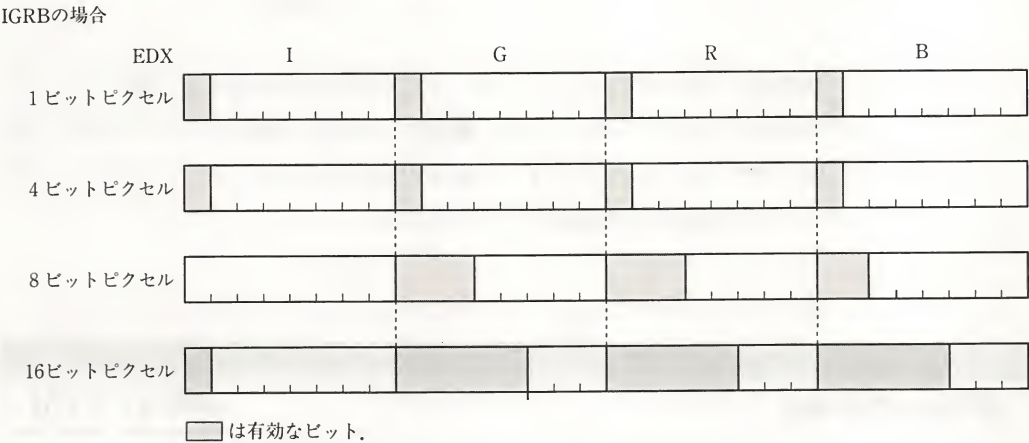
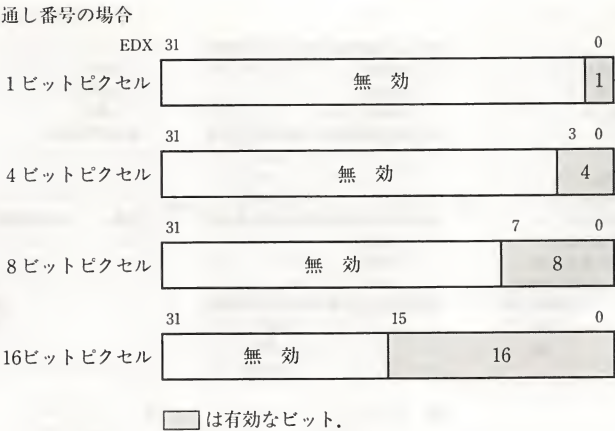
AL =設定モード(0：通し番号，1：IGRB)

EDX =設定色

リターン AH =00H(正常終了時)

説　明 マウスカーソル(システム形状と単色の場合)の表示色を色識別番号で指定します.

設定モードが0のときは，通し番号で，1のときはIGRBで指定します.

EDXの形式を示します.

マウス	40H
タイルパターンの設定	**機能コード10H**

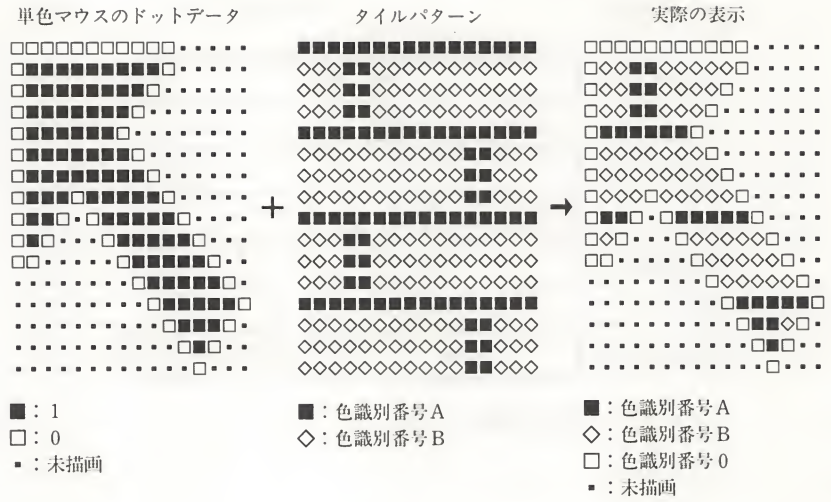
エントリ　AH　＝10H
BH　＝水平サイズ(1～4)
BL　＝垂直サイズ(1～32)
DS：ESI　＝タイルパターンデータのアドレス

リターン　AH　＝00H(正常終了時)

説　明

単色マウスカーソルのタイルパターンを設定します.

水平サイズは，1，2，3，4のいずれか(設定されるサイズはそれぞれ，8，16，24，32)，垂直サイズは1～32の整数で指定します.

タイルパターンの例(16×16)を示します.

この例は，単色マウスカーソルをシステム形状と同じ形にした場合です．図中の単色マウスのドットデータの，■は1，□は0を表します．この場合，表示されるカーソルは，ドットデータが1の部分がタイルパターンで設定した色となり，□の部分は色識別番号0となります．

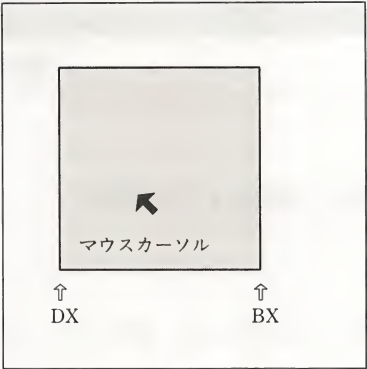
マウス　40H

水平消去範囲指定　機能コード11H

エントリ	AH	＝11H
	AL	＝マウス消去範囲の解除／設定(0：解除，1：設定)
	DX	＝最小水平座標
	BX	＝最大水平座標
リターン	AH	＝00H(正常終了時)

説　明

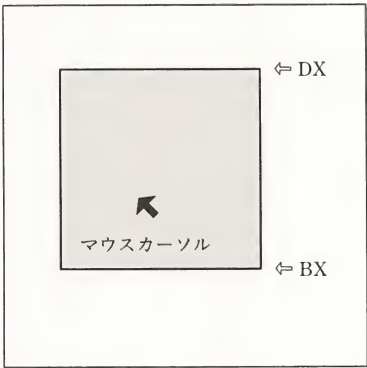
マウスカーソルの水平消去範囲の設定と解除を行います．

設定された範囲にマウスカーソルが入ると，マウスカーソルの表示，非表示にかかわらず，マウスカーソルは表示されません．

仮想画面枠

マウス	40H
垂直消去範囲指定	機能コード12H

エントリ　AH　＝12H
AL　＝マウス消去範囲の解除／設定(0：解除，1：設定)
DX　＝最小垂直座標
BX　＝最大垂直座標

リターン　AH　＝00H(正常終了時)

説　明　マウスカーソルの垂直消去範囲の設定と解除を行います．
設定された範囲にマウスカーソルが入ると，マウスカーソルの表示，非表示にかかわらず，マウスカーソルは表示されません．

仮想画面枠

マウス	40H
ボタン左右入れ換え状態の設定	**機能コード13H**

エントリ	AH	＝13H
	AL	＝左右入れ換え状態(0：通常，1：入れ換え)
リターン	AH	＝00H(正常終了)

説　明

マウスボタンの左右入れ換えをしたいとき1，通常に戻したいとき0を設定します．

初期状態は0(通常)が設定されています．

マウス	40H
加速度検出状態の設定	**機能コード14H**

エントリ	AH	＝14H
	AL	＝加速度検出状態(0：無効，1：有効)
リターン	AH	＝00H(正常終了)

説　明

マウスの加速度を検出したいとき1，無効にしたいとき0を設定します．

初期状態は1(有効)が設定されています．

マウス　40H

解像度ハンドルによるマウスの仮想画面設定　機能コード 15H

エントリ　AH　＝15H
AL　＝解像度ハンドル

リターン　AH　＝00H（正常終了時）
－1（エラー）

説　明　システム情報 BIOS で取得した解像度ハンドルを使って，マウスの仮想画面の設定を行うオペレーションです．ただし，画面の表示制御は含まれないので，必要ならばユーザーがグラフィックス BIOS で行わなければなりません．

仮想画面の設定内容は次のとおりです．

マウスカーソル位置	画面中央
マウスカーソル形状	システム形状
マウスカーソル移動範囲	画面全体

第 5 章

フォントBIOS

FMTOWNSには，ANK文字(グラフィックキャラクタを含む)とJIS第1，第2水準の漢字フォントが内蔵されており，フォントBIOSを使って読み出すことができます．

この章では，このフォントBIOSについて解説します．

5.1 フォントBIOS一覧

表II-5-1に，フォントBIOS一覧を示します．

なお，フォントBIOSでサポートしているフォントの種類は

ANK　(8×8ドット
　　　8×16ドット)

漢字　(16×16ドット)

の3種類です．

ANKは1バイトコードで読み出します．また，漢字はJISコードで読み出します．フォントBIOSでは，シフトJISコードとJISコードの相互の変換もサポートしているので，シフトJISコードを元にして漢字のフォントを読み出すことが容易にできます．

▼表II-5-1　フォントBIOS一覧

機能名称	機能コード
ANKフォントの読み出し	00H
漢字フォントの読み出し	01H
シフトJISからJISへの変換	02H
JISからシフトJISへの変換	03H

5.2 フォントBIOSリファレンス

フォント BIOS について個別に詳しく解説します．

フォント	A0H
ANK フォントの読み出し	機能コード00H

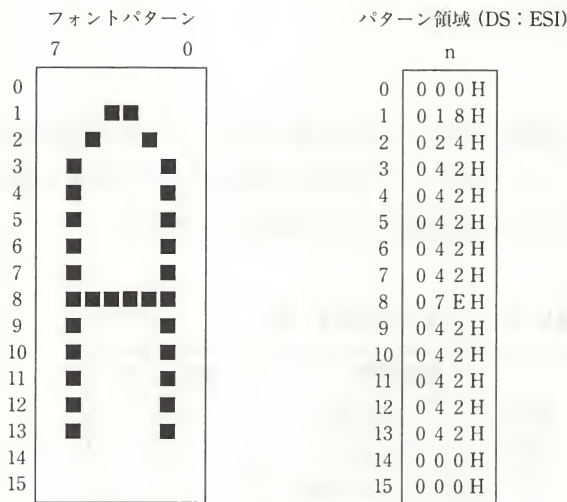
エントリ

AH	＝00H
AL	＝機能番号(0：アドレスを求める，1：フォントを転送する)
DH	＝横ドット数(8)
DL	＝縦ドット数(8または16)
BL	＝ANK 文字コード
DS：ESI	＝フォント転送先アドレス(AL＝1のとき)

リターン

AH	＝00H(正常終了時)
DS：ESI	＝フォントの ROM 内アドレス(AL＝0のとき)

説　明

ANK 文字フォントの ROM 内アドレス，または，フォントを読み出します．ANK フォントの例を示します．

(パターンサイズ8×16の場合：パターン格納領域サイズ16バイト)

フォント　　　　　　　　　　　　　　　　　　　　A0H

漢字フォントの読み出し　　　　　　　　　　機能コード01H

エントリ

AH	＝01H
AL	＝機能番号（0：アドレスを求める，1：フォントを転送する）
DH	＝横ドット数(16)
DL	＝縦ドット数(16)
BX	＝漢字コード(JIS)
DS:ESI	＝フォント転送先アドレス(AL＝1のとき)

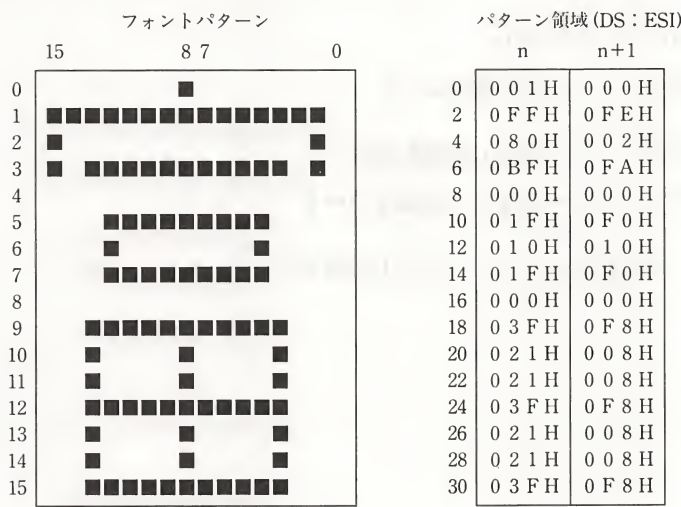
リターン

AH	＝00H（正常終了時）
DS:ESI	＝フォントのROM内アドレス(AL＝0のとき)

説　明

漢字フォントのROM内アドレス，または，フォントを読み出します．
漢字フォントの例を示します．

パターン領域(DS：ESI)

	n	n+1
0	001H	000H
2	0FFH	0FEH
4	080H	002H
6	0BFH	0FAH
8	000H	000H
10	01FH	0F0H
12	010H	010H
14	01FH	0F0H
16	000H	000H
18	03FH	0F8H
20	021H	008H
22	021H	008H
24	03FH	0F8H
26	021H	008H
28	021H	008H
30	03FH	0F8H

（パターンサイズ16×16の場合：パターン格納領域サイズ32バイト）

フォント	A0H
シフトJISからJISへの変換	**機能コード02H**

エントリ	AH	＝02H
	BX	＝シフトJIS漢字コード
リターン	AH	＝00H (正常終了時)
	BX	＝JIS漢字コード

説　明

シフトJIS漢字コードをJIS漢字コードに変換します.

「漢字フォントの読み出し(機能コード01H)」など, JIS漢字コードが必要なオペレーションを行う前に, このオペレーションを使用して変換を行います.

フォント	A0H
JISからシフトJISへの変換	**機能コード03H**

エントリ	AH	＝03H
	BX	＝JIS漢字コード
リターン	AH	＝00H (正常終了時)
	BX	＝シフトJIS漢字コード

説　明

JIS漢字コードをシフトJIS漢字コードに変換します.

第6章

サウンドBIOS

サウンドBIOSでは，FM音源やPCM音源による演奏や，PCMサンプリングを行うことができます．また，電子ボリューム，TOWNSマウス，TOWNSパッドの制御もサポートしています．

この章では，このサウンドBIOSについて解説します．

6.1 サウンドBIOSの位置づけ

FMTOWNSには，FM音源LSI(YM2612)とPCM音源LSI(RF5C68)が搭載されていますが，それらをコントロールするソフトウェア(ドライバ)は図II-6-1のようなレベルがあります．

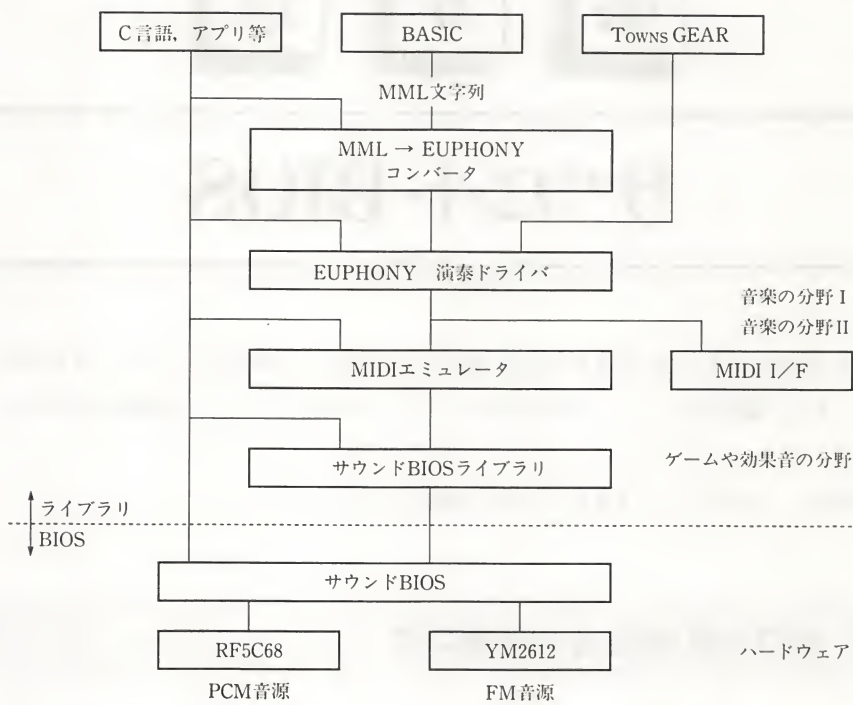
ユーザーはどのレベルからでも音源LSIを動作させることができますが，最もハードウェア寄りのものが，サウンドBIOSです．

サウンドBIOS上には，サウンドBIOSライブラリ，MIDIエミュレータ，EUPHONY(MIDI対応のソフトウェア)演奏ドライバなどがあります．

MIDIエミュレータは，FMTOWNSの各音源をMIDI機器としてみなして制御できるようにしているものです．これにより，MIDIカードに対応したアプリケーションを用いて，容易に内蔵音源に対応することができます．

EUPHONY演奏ドライバは，EUPHONY互換データ形式を採用しています．このデータ形式は，従来のMML(Music Macro Language)より，データの分解能が細かく，トラック数も大幅に増えており，よりきめ細かく，かつ重厚なサウンドを作り出すことができます．また，データの作成には，EUPHONYを用いることができるので，データの作成効率は格段に向上します．

また，従来のMML文字列をEUPHONY形式に変換するコンバータが用意されているので，BASICなどのMML文字列を利用して，音源を制御することも可能になっています．

▼図II-6-1　サウンドBIOSの位置づけ

6.2　サウンドBIOS一覧

サウンドBIOSは，次の4種類に分類することができます．

1．FM音源／PCM音源共通のもの
2．FM音源のみのもの
3．PCM音源のみのもの
4．TOWNSパッド，TOWNSマウス制御
5．電子ボリューム制御

サウンドBIOSの機能一覧を表II-7-1に示します．

▼表II-6-1　サウンドBIOS一覧

機能名称	機能コード	機能名称	機能コード
ドライバの初期化	00H	サウンドの削除	23H
キーON	01H	PCMサンプリング開始	24H
キーOFF	02H	音声モードPCM再生	25H
出力先指定	03H	PCMサンプリング中断	26H
音色変更	04H	音声モードPCM再生中断	27H
音色データの書き込み	05H	音声モードPCM再生状態参照	28H
音色データの読み出し	06H	PCM音源の強制停止	29H
ピッチベンド	07H	PCMメモリ→メインメモリ転送	2AH
ボリューム変更	08H	PCMメモリ→PCMメモリ転送	2BH
発音の強制停止	09H	PCMメモリ転送2	2CH
音声モードPCM再生アドレスの読み取り	0AH	高品位音声モードPCM再生	2EH
FM音源ステータスレジスタの読み出し	10H	FM音源のみの初期化	30H
FM音源1バイト出力	11H	FM音源レジスタの書き込み	31H
FM音源1バイト入力	12H	パッド入力1	40H
FM音源レジスタの書き込み	13H	パッド入力2	41H
FM音源レジスタの読み出し	14H	パッド出力	42H
タイマAコントロール1	15H	電子ボリューム設定	43H
タイマBコントロール1	16H	電子ボリューム初期化	44H
タイマAコントロール2	17H	電子ボリューム設定読み出し	45H
タイマBコントロール2	18H	電子ボリュームミュート	46H
ハードLFOの設定	19H	電子ボリューム全ミュート	49H
PCMメモリ転送	20H	エンベロープ割り込みエントリ	50H
音声モードチャネルの設定	21H	音声モード割り込みエントリ	51H
サウンドの登録	22H		

6.3　サウンドBIOSの基本機能と用語

ここでは，サウンドBIOSの基本的な機能と用語について解説します．

● PCM音源の楽器モードと音声モード

サウンドBIOSでは，PCM音源による再生を，楽器モードと音声モードの2種類の方法で行います．

楽器モードは，PCM音源を音楽演奏の際の楽器のように使用することを目的としたものです．波形メモリに格納したデータをもとに，ボリューム，ピッチ(再生速度)，ループ(繰り返し)，エンベロープなどのデータを付加して再生します．

音声モードは，音声メッセージを再生することを目的としています．ユーザーメモリ内に波形データを展開し，これを波形メモリに転送しながら再生を行います．このモードでは，ボリュームとピッチとループの制御はできますが，エンベロープ制御はできません．

●楽器モードのデータ構造

楽器の音色は，低音から高音まで同じ音色とは限りません．例えば，ピアノの場合，低音部では筐体の共鳴音が音色に加わり，高音部では弦の部分の振動のみが音色としてとらえられます．また，中音部では弦の振動に微妙な共鳴音が加わります．このように，1つの楽器といえども音色は必ずしも，一様ではありません．

そこで，音域ごとに音をサンプリングし，それぞれが分担する音域を決めて再生すると，リアルな音を再現することができるようになります．

音域ごとのデータをサウンドデータと呼びます．また，サウンドデータをどの音域に割り当てるか，それぞれのサウンドをどのようなエンベロープで演奏するかなどを示すデータをインスツルメントデータといいます．サウンドデータとインスツルメントデータの組み合せによって，さまざまな音が再現できます

サウンドデータ

サウンドデータは，サンプリングした波形データにヘッダ情報として，サンプリング周波数，サンプリングレートなどを付け加えたものです．

表II-6-2にサウンドデータの形式を示します．

▼表II-6-2　サウンドデータの形式

	サイズ	内　容	説　明
0	8	サウンドネーム	8文字の名前
8	DW	サウンドID	BIOS内のサウンド管理識別用ID
12	DW	データ幅	サンプリングデータの総バイト数
16	DW	ループポイント	ループの開始点
20	DW	ループレングス	ループの長さ（0のときはループなし）
24	W	サンプリング周波数	サンプリング時のサンプリング周波数
26	W	原音の補正値	サンプリング周波数に対する加減算値
28	B	原音の音階	サウンドデータの基本音階
29	B	予約済	0にする
30	W	予約済	0にする
32	データ幅	波形データ	PCMデータ

波形データの長さは最大64KBまでです．波形データは，量子化ビット数が8ビットで，最上位ビットは符号です．また，FFHはループストップデータであり，通常のデータではありません．詳しくは，第1部の「第5章　オーディオシステム」を参照してください．なお，サウンドデータのサンプリング周波数に書く値(F)は，次の式によって求められます．

$$F = freq(KHz) \times 62H$$

サウンドデータの先頭の32バイトの情報は，登録時(「サウンドの登録(機能コード22H)」)にBIOSのサウンド管理用領域にコピーされ，発音のためのデータとして使用されます．

サウンドデータは複数のインスツルメントから参照することができます．

インスツルメントデータ

インスツルメントデータは，どの音域にどのサウンドデータを使用するか，それぞれのサウンドデータをどのようなエンベロープで鳴らすかを示すものです．

表II-6-3にインスツルメントデータの情報のフォーマットを示します．

▼表II-6-3　インスツルメントデータの形式

	サイズ	内　容	説　明
0	8	プログラムネーム	8文字の名前
8	8	予約済	
16	W	スプリット1	サウンド1の音階の上限
18	W	スプリット2	サウンド2の音階の上限
30	W	スプリット8	サウンド8の音階の上限
32	DW	サウンドID1	スプリット1に使用するサウンドID
36	DW	サウンドID2	スプリット2に使用するサウンドID
60	DW	サウンドID8	スプリット8に使用するサウンドID
64	8	エンベロープ1	スプリット1のエンベロープ
72	8	エンベロープ2	スプリット2のエンベロープ
120	8	エンベロープ8	スプリット8のエンベロープ

このように1つのインスツルメントデータには，最大8個のサウンドデータを登録することができます．また，各サウンドごとにエンベロープを設定できます．

楽器モードのエンベロープ

楽器モードでは，サウンドごとにエンベロープの設定ができます．これにより，比較的少ないデータ量で，リアルな音の再現が可能になります．

エンベロープデータの形式を表II-6-4に示します．また，エンベロープの意味を図II-6-4に示します．

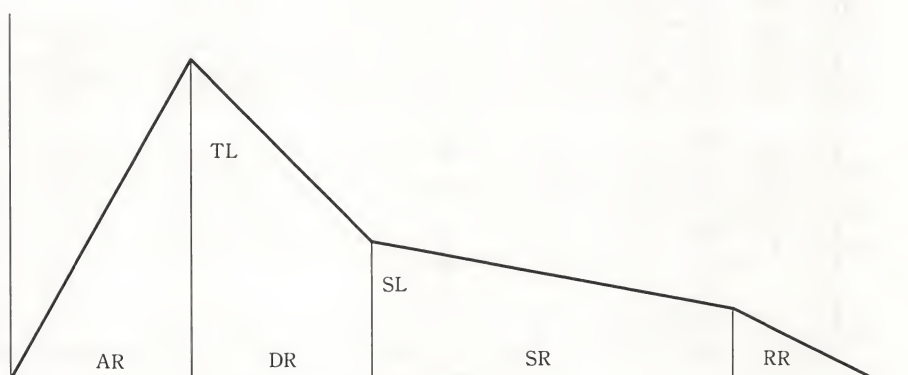
エンベロープをコントロールするためには，各アプリケーションなどで，10msごとに更新ルーチンを呼ばなければなりません．ただし，Cのサウンド BIOS ライブラリを使用した場合は，SND_init を用いて初期化すれば，自動的に割り込みルーチンが登録されるのでこの必要はありません．

ゲームなどの効果音として PCM 音源を使用したいが，割り込み処理を毎回呼びたくないというときは，エンベロープデータを次のように設定します．そうするとエンベロープ機能は使用できませんが，割り込み処理を登録せずに PCM 音源が利用できます．

TL＝0～127，AR＝0，DR＝127，SL＝0～127，SR＝127，RR =127

▼表II-6-4　エンベロープデータの形式

	サイズ	内　容	説　明	
0	B	トータルレベル	最大音量	0～127
1	B	アタックレート	アタックの増加レート	0～127
2	B	ディケイレート	アタック後の減衰レート	0～127
3	B	サスティンレベル	減衰レベル	0～127
4	B	サスティンレート	減衰レート	0～127
5	B	リリースレート	キーOFF時の減衰レート	0～127
6	B	ルートキー	各スプリット毎のルートに対するオフセット	−128～127
7	B	予約済	0にする	

▼図II-6-2　エンベロープの意味

TL　トータルレベル　最大音量，127で最大
AR　アタックレート　最大音量に達するまでの時間，0で最も速い
DR　ディケイレート　サスティンレベルに達するまでの時間
SL　サスティンレベル　持続音の音量
SR　サスティンレート　持続音の減衰する時間
RR　リリースレート　キーオフ後の余韻の時間

楽器モードのバンク

インスツルメントデータとサウンドデータをひとまとめにしてファイルに格納することができます．その形式をバンクといいます．バンクの形式を表II-6-5に示します．

1つのバンクにはサウンドデータ128個とインスツルメントデータ32個を格納できます．インスツルメントデータに登録する8個のサウンドデータは128個の中から選択できます．ただし，サウンドデータは，128個すべてを書き込む必要はありません．バンクデータのファイルの拡張子は〝.PMB〞で表します．

楽器モードで発音させるためのデータの準備

インスツルメントデータを，「音色データの書き込み(機能コード05H)」で音色格納領域(サウンドBIOSの作業領域)に転送し，「音色変更(機能コード04H)」でPCM音源が発音する音

▼表II-6-5　楽器モードのバンクデータの形式

サイズ	内　容	説　明
8	バンクネーム	8文字までの名前
128	インスツルメント1	
128	インスツルメント2	
128	インスツルメント3	
128	インスツルメント32	
不定	サウンド1	
不定	サウンド2	
不定	サウンド3	
不定	サウンド128	

色データ番号を指定します．

インスツルメントに登録したサウンドデータ中の波形データを，「サウンドの登録(機能コード22H)」で，波形メモリに転送します．このときサウンドデータのヘッダの32バイトは，サウンドBIOSの作業領域に転送されます．

●音声モードのデータ構造

音声モードのデータは，サウンドデータのみです．インスツルメントはありません．

データは楽器モードのサウンドデータと同様の形式(表II-6-6)になります．サウンドデータの長さは，メモリの許す限り設定できます．

音声モードのデータのファイルの拡張子は〝.SND〞です．

▼表II-6-6　音声モードのサウンドデータの形式

	サイズ	内　容	説　明
0	8	サウンドネーム	8文字の名前
8	DW	サウンドID	BIOS内のサウンド管理識別用ID
12	DW	データ幅	サンプリングデータの総バイト数
16	DW	ループポイント	ループの開始点
20	DW	ループレングス	ループの長さ　(0のときはループなし)
24	W	サンプリング周波数	サンプリング時のサンプリング周波数
26	W	原音の補正値	サンプリング周波数に対する加減算値
28	B	原音の音階	サウンドデータの基本音階
29	B	予約済	0にする
30	W	予約済	0にする
32	データ幅	波形データ	PCMデータ

音声モードで発音させるためのデータの準備

サウンドデータ中の波形データは，発音時に4KBずつ波形メモリに転送されて，随時再生されます．したがって，音声モードの場合は，「音声モードチャネルの設定(機能コード21H)」で音声モードに指定するチャネルを設定し，割り込みで「音声モード割り込みエントリ(機能コード51H)」を呼び，「音声モードPCM再生(機能コード25H)」を実行すれば再生ができます．

● FM音源のデータ構造

FM音源では，各インスツルメントが，音域ごとにサウンドデータを持ちません．インスツルメントがそのまま音色データとなっています．1つのインスツルメントのデータの長さは48バイトです．

FM音源のインスツルメントデータの形式を表II-6-7に示します．

各パラメータには，4つのスロットのデータを，1，3，2，4の順で格納します．

▼表II-6-7　FM音源のインスツルメントデータの形式

	サイズ	内　容	説　明
0	8	プログラムネーム	8文字までの名前
8	4	DT1，MULT1	ディチューン，マルチプル
12	4	TL	トータルレベル
16	4	KS，AR	キースケール，アタックレート
20	4	AMON，DR	ディケイレート
24	4	SR	サスティンレート
28	4	SL，RR	サスティンレベル，リリースレート
32	B	FB，CNCT	フィードバック，コネクト
33	B	LR，AMS，PMS	パン，LFO
34	14	予約済	0にする

FM音源用データを格納するファイルは，インスツルメントの集合体であるバンクのデータ形式をとります．そのデータ形式を表II-6-8に示します．

FM音源バンクデータのファイルの拡張子は，〝.FMB″となっています．

▼表II-6-8　FM音源のバンクデータ

サイズ	内　容	説　明
8	バンクネーム	8文字までの名前
48	インスツルメント1	1番目の音色データ
48	インスツルメント2	2番目の音色データ
48	インスツルメント128	128番目の音色データ

FM音源で発音させるためのデータの準備

インスツルメントデータを，「音色データの書き込み(機能コード05H)」で音色格納領域(サウンドBIOSの作業領域)に転送し，そのうちの音色データを「音色変更(機能コード04H)」でFM音源LSIに書き込みます．

6.4　サウンドBIOSオペレーションの共通事項

ここでは，サウンドBIOSの各オペレーションに共通する事項について解説します．

●作業領域

作業領域(ワーク)を必要とするBIOSを呼ぶときは，作業領域(16KB)の先頭アドレスをGS：EDIに入れておかなければなりません．

●FM音源レジスタのアクセス

サウンドBIOSを使って，FM音源のレジスタに対して読み書きすることができます．

各レジスタの意味については，「第1部第5章　オーディオシステム」を参照してください．

●チャネルと音源の対応

サウンドBIOSのオペレーションでは，チャネルを指定する場合があります．表II-6-9に，チャネルと音源との関係を示します．指定する際にはこの表に従って番号を選択してください．

▼表II-6-9　チャネルの対応

チャネル	音源種別	チャネル	音源種別
0	FM音源　ch1	64	PCM音源　ch1
1	FM音源　ch2	65	PCM音源　ch2
2	FM音源　ch3	66	PCM音源　ch3
3	FM音源　ch4	67	PCM音源　ch4
4	FM音源　ch5	68	PCM音源　ch5
5	FM音源　ch6	69	PCM音源　ch6
6～15	FM音源拡張用	70	PCM音源　ch7
16～31	拡張用	71	PCM音源　ch8
32～63	リズム音源等拡張用	72～95	PCM音源拡張用
		96～127	その他の音源拡張用

拡張用を除き，FM音源とPCM音源のチャネルはハードウェアのチャネルと対応しています．

●エラーコードについて

サウンド以外のBIOSでは，リターンコードはAHレジスタに返されますが，サウンドBIOSの場合はALに返されます．返される値と意味を表II-6-10に示します．

▼表II-6-10　ALに返される値の意味

AL	意　味
00H	正常終了
01H	チャネル番号が異常
02H	キーON中
03H	パラメータエラー
04H	未定義ファンクション
05H	波形メモリ不足
06H	サウンド内にデータ長が存在しない
07H	音声モードに使えないチャネル
08H	スプリットの上限音階が小さすぎる
09H	サウンドIDが見つからない
0AH	サウンドの二重登録
0BH	音声モードの上限周波数を超えた
0CH	サンプリングを強制停止した
0DH	サウンドデータのヘッダが異常である

6.5　サウンドBIOSリファレンス

サウンドBIOSについて個別に詳しく解説します．

サウンド	80H
ドライバの初期化	機能コード00H

エントリ　AH　＝00H

GS：EDI　＝作業領域のアドレス(16KB必要)

リターン　AL　＝00H(正常終了)

説　明　サウンド関係のハードウェアとBIOSを初期化します．

BIOSには16KB(16384バイト)の作業領域(ワーク)が必要であり，GS：EDIにはその先頭アドレスを指定します．

初期化では，以下の処理が実行されます．

- FM 音源 LSI の初期化
- FM 音源の発音の停止
- FM 音源タイマの停止
- PCM 音源の初期化
- PCM 音源の発音の停止
- 波形メモリの初期化
- 電子ボリュームの初期化(すべての入力要素にミュートがかかる)
- ワークの初期化

項　目		機能コード	初期値
FM音源	キー OFF	02H	OFF
	出力先指定	03H	中央
	音色変更	04H	デフォルト音色(ELEPIANO)設定
	音色データの書き込み	05H	デフォルト音色(ELEPIANO)書き込み
	ピッチベンド	07H	ピッチリセット
	ボリューム変更	08H	最大値設定
PCM音源	キー OFF	02H	OFF
	出力先指定	03H	中央
	音色変更	04H	無音音色設定
	音色データの書き込み	05H	無音音色書き込み
	ピッチベンド	07H	ピッチリセット
	ボリューム変更	08H	最大値設定

サウンド　80H

キー ON　機能コード01H

エントリ　AH　＝01H
BL　＝チャネル番号(0～127)
DH　＝音程(0～127)
DL　＝音量(1～127)

リターン　AL　＝00H (正常終了)
01H (チャネル番号エラー)
02H (キー ON 中)
03H (パラメータエラー)

説　明

音源(チャネル)の発音を開始します．音程と音量の設定はMIDIに準拠しており，それぞれ127までの値が設定できます．MIDI規格では，音量0がキーOFFを意味するため，このオペレーションでは0を含めていません．

音程の対応を示します．

音階＼オクターブ		01 (#)		02 (#)		03 (#)		04 (#)		05 (#)		06 (#)		07 (#)		08 (#)	
C	ド	24	(25)	36	(37)	48	(49)	60	(61)	72	(73)	84	(85)	96	(97)	108	(109)
D	レ	26	(27)	38	(39)	50	(51)	62	(63)	74	(75)	86	(87)	98	(99)	110	(111)
E	ミ	28		40		52		64		76		88		100		112	
F	ファ	29	(30)	41	(42)	53	(54)	65	(66)	77	(78)	89	(90)	101	(102)	113	(114)
G	ソ	31	(32)	43	(44)	55	(56)	67	(68)	79	(80)	91	(92)	103	(104)	115	(116)
A	ラ	33	(34)	45	(46)	57	(58)	69	(70)	81	(82)	93	(94)	105	(106)	117	(118)
B	シ	35		47		59		71		83		95		107		119	

サウンド　80H
キーOFF　機能コード02H

エントリ

AH　＝02H

BL　＝チャネル番号(0～127)

リターン

AL　＝00H(正常終了)

　　01H(チャネル番号エラー)

説　明

音源(チャネル)の発音停止を指示します．このオペレーションを実行すると，音源はリリースレートの減衰特性に従って発音を停止します．このため，リリースレートの設定によっては，すぐに発音を停止しないこともあります．

強制的に停止させたい場合には，「発音の強制停止(機能コード09H)」を使用します．

サウンド	80H
出力先指定	**機能コード03H**

エントリ	AH	＝03H
	BL	＝チャネル番号(0～127)
	DL	＝パンポット(0～127)
リターン	AL	＝00H(正常終了)
		01H(チャネル番号エラー)

説　明

各音源の左右の出力先を設定します.

DLに設定する値は，0が左のみ，64が中間，127が右のみの出力です.

0	左音声ラインのみに出力する
⋮	
64	左右均等に出力する
⋮	
127	右音声ラインのみに出力する

ただし，0～127の範囲で値が設定できるのはPCM音源のみで，FM音源の場合は0，64，127の3種類しか指定できません.

このオペレーションは，PCM音源(楽器モード，音声モード)，FM音源のすべてで使用できます.

サウンド	80H
音色変更	**機能コード04H**

エントリ	AH	＝04H
	BL	＝チャネル番号(0～127)
	DH	＝音色番号(0～127)
リターン	AL	＝00H(正常終了)
		01H(チャネル番号エラー)
		03H(パラメータエラー)

説　明

FM音源とPCM音源(楽器モード)において，「音色データの書き込み(機能コード05H)」で音色格納領域に格納したインスツルメントデータのうち，音色の設定に必要なデータを音源に設定します.

サウンド　80H
音色データの書き込み　機能コード05H

エントリ	AH	＝05H
	BL	＝チャネル番号(0～127)
	DH	＝音色番号(0～127)
	DS:ESI	＝インスツルメントデータのアドレス
リターン	AL	＝00H(正常終了)
		01H(チャネル番号エラー)
		03H(パラメータエラー)

説　明

FM音源とPCM音源(楽器モード)において，音色番号で指定された音色格納領域に，DS：ESIで示すアドレスに準備したインスツルメントデータを転送します.

各チャネルの音色を変更するには，「音色変更(機能コード04H)」を使って，音色格納領域のデータを音源LSIに転送します.

サウンド　80H
音色データの読み出し　機能コード06H

エントリ	AH	＝06H
	BL	＝チャネル番号(0～127)
	DH	＝音色番号(0～127)
	DS:ESI	＝インスツルメントデータのアドレス
リターン	AL	＝00H(正常終了)
		01H(チャネル番号エラー)
		03H(パラメータエラー)

説　明

FM音源とPCM音源(楽器モード)において，指定の音色番号に登録されているインスツルメントデータを，DS：ESIで示すアドレスの領域に転送します.

サウンド	80H
ピッチベンド	**機能コード07H**

エントリ　AH　=07H
　　　　　BL　=チャネル番号(0～127)
　　　　　DX　=ピッチ(−8192～+8191)

リターン　AL　=00H(正常終了)
　　　　　　　01H(チャネル番号エラー)
　　　　　　　03H(パラメータエラー)
　　　　　　　0BH(音声モードの上限周波数を越えた)

説　明　音源(チャネル)のピッチを設定します．音程を微調整するときに使用します．

DX に指定するピッチの値の範囲は，−8192から+8191までです．

正常なピッチを0として，最大値～最小値までで，2オクターブの範囲を持ちます．マイナスの場合は音程が下がり，プラスの場合は上がります．

ピッチの変更はピッチリセット(DX=0)を行うまで有効で，ピッチ変更以降のキーON と音声出力では，それぞれに指定した音程+ピッチ変更値でキーON や音声出力が行われるので注意してください．

また，PCM 音源の音声モードでは，サウンドデータのサンプリング周波数が高い場合，ピッチの設定値まで音程が上がらない場合があります．

このオペレーションは，PCM 音源(楽器モード，音声モード)，FM 音源のすべてで使用できます．

サウンド	80H
ボリューム変更	**機能コード08H**

エントリ　AH　=08H
　　　　　BL　=チャネル番号(0～127)
　　　　　DL　=ボリューム(0～127)

リターン　AL　=00H(正常終了)
　　　　　　　01H(チャネル番号エラー)
　　　　　　　03H(パラメータエラー)

説　明　各音源(チャネル)の音量を設定します．音量の設定範囲は，0(無音)から127(最大)の範囲です．デフォルト値は127です．

このオペレーションは，PCM 音源(楽器モード，音声モード)，FM 音源のすべてで使用できます．

サウンド	80H
発音の強制停止	**機能コード09H**

エントリ　AH　=09H
BL　=チャネル番号(0～127)

リターン　AL　=00H(正常終了)
01H(チャネル番号エラー)
03H(パラメータエラー)

説　明

各音源(チャネル)の発音を強制的に停止します.

リリース中でも強制的に停止されるので，通常のキーOFFの代わりに使用すると不自然な発音となります.

このオペレーションは，PCM音源(楽器モード，音声モード)，FM音源のすべてで使用できます.

サウンド	80H
音声モードPCM再生アドレスの読み取り	**機能コード0AH**

エントリ　AH　=0AH
BL　=チャネル番号 (64～71)
EDX　=0
DS：ESI　=読み取り値転送先アドレス

リターン　AL　=00H (正常終了)
01H (チャネル番号エラー)
03H (パラメータエラー)

説　明

指定されたチャネルの，音声モードでの再生アドレス現在値を，転送先アドレスで示されたメモリに取得します.

転送された内容は，次のような形式になっています.

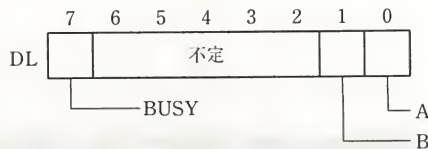
(DS：ESI)

0	DW	PCM再生アドレス
4	W	PCM再生セレクタ

サウンド	80H
FM音源ステータスレジスタの読み出し	機能コード10H

エントリ　AH　＝10H

リターン　DL　＝ステータス
AL　＝00H (正常終了)
　　03H (パラメータエラー)

説　明　FM音源のステータスレジスタの値をDLに返します．
このオペレーションはワークが不要です．
DLの形式を示します．

サウンド	80H
FM音源1バイト出力	機能コード11H

エントリ　AH　＝11H
BH　＝バンク番号(0，1)
DH　＝レジスタ番号
DL　＝データ

リターン　AL　＝00H (正常終了)
　　03H (パラメータエラー)

説　明　FM音源のレジスタに1バイトのデータを書き込みます．

BHのバンク番号には，チャネル0〜2用のレジスタに書き込む場合は0を，チャネル3〜5用のレジスタに書き込む場合は1を指定します．

このオペレーションは，特殊な効果音を作る場合などに，直接FM音源レジスタへの書き込みをするときに使用します．なお，普通にインスツルメントデータを使用してFM音源を鳴らす際には，FM音源レジスタへの書き込みは，「音色変更(機能コード04H)」を使用します．

サウンド	80H
FM 音源1バイト入力	機能コード12H

エントリ
AH　=12H
BH　=バンク番号(0, 1)
DH　=レジスタ番号

リターン
DL　=データ
AL　=00H(正常終了)
　　03H(パラメータエラー)

説　明
FM音源のレジスタの値を読み込みます．ただし，現在のFMTOWNSで使用されているFM音源のレジスタは，書き込み専用で，読み取れるレジスタは存在しません．

サウンド	80H
FM 音源レジスタの書き込み	機能コード13H

エントリ
AH　=13H
BH　=バンク番号(0, 1)
DH　=レジスタアドレス(0～255)
DL　=データ

リターン
AL　=00H(正常終了)
　　03H(パラメータエラー)

説　明
FM音源レジスタの複写領域に指定データを書き込むオペレーションです．
BHのバンク番号には，チャネル0～2用のレジスタに書き込む場合は0を，チャネル3～5用のレジスタに書き込む場合は1を指定します．

サウンド　80H

FM音源レジスタの読み出し　機能コード14H

エントリ　AH　＝14H
BH　＝バンク番号(0，1)
DH　＝レジスタアドレス(0～255)

リターン　DL　＝データ
AL　＝00H(正常終了)
03H(パラメータエラー)

説　明

「FM音源レジスタの書き込み(機能コード13H)」でFM音源レジスタ複写領域に書き込んだデータを読み出します．

BHのバンク番号には，チャネル0～2用のレジスタから読み出す場合は0を，チャネル3～5用のレジスタから読み出す場合は1を指定します．

サウンド　80H

タイマAコントロール1　機能コード15H

エントリ　AH　＝15H
BL　＝スイッチ(0，255)
CX　＝カウンタ(0～1023)

リターン　AL　＝00H(正常終了)
03H(パラメータエラー)

説　明

タイマAのカウント動作を制御します．

スイッチの値が0以外のときは，CXにセットされたカウント値をFM音源に設定し，カウンタをスタートさせます．スイッチの値が0のときは，タイマを停止させ，ステータスフラグをリセットします．つまり，タイマを停止するときは，スイッチを0にしてこのオペレーションを実行します．

カウント時間は0が最大で，1023が最小値です．例えば，0の場合は0からインクリメントして1023までカウントします．1023をオーバーした時点が，タイマ割り込みを生ずるタイミングとなります．

カウント値(N)と時間間隔(T)との関係は次の式で表されます．

$$T\,(\mathrm{ms}) = 12 \times (1024 - N) \times 12 / 8000\,(\mathrm{KHz})$$

サウンド　80H

タイマBコントロール1　機能コード16H

エントリ	AH	=16H
	BL	=スイッチ(0, 255)
	CX	=カウンタ(0～255)
リターン	AL	=00H(正常終了)
		03H(パラメータエラー)

説　明

タイマBのカウント動作を制御します.

スイッチの値が0以外のときは, CXにセットされたカウント値をFM音源に設定し, カウンタをスタートさせます. スイッチが0のときは, タイマを停止させステータスフラグをリセットします. つまり, タイマを停止するときは, スイッチを0にしてこのオペレーションを実行します.

カウント時間は0が最大で, 255が最小値です. 例えば, 0の場合は0からインクリメントして255までカウントします. 255をオーバーした時点が, タイマ割り込みを生ずるタイミングとなります.

カウント値(N)と時間間隔(T)との関係は次式で表されます.

$$T(\mathrm{ms}) = 192 \times (256 - N) \times 12 / 8000(\mathrm{KHz})$$

サウンド	80H
タイマAコントロール2	**機能コード17H**

エントリ　AH　＝17H

リターン　AL　＝00H(正常終了)
03H(パラメータエラー)

説　明　タイマAを，「タイマAコントロール1(機能コード15H)」で停止させた後，再スタートさせるときに使用します．

カウント値は「タイマAコントロール1(機能コード15H)」の設定が引き継がれます．

サウンド	80H
タイマBコントロール2	**機能コード18H**

エントリ　AH　＝18H

リターン　AL　＝00H(正常終了)
03H(パラメータエラー)

説　明　タイマBを，「タイマBコントロール1(機能コード16H)」で停止させた後，再スタートさせるときに使用します．

カウント値は「タイマBコントロール1(機能コード16H)」の設定が引き継がれます．

サウンド	80H
ハードLFOの設定	**機能コード19H**

エントリ AH =19H
DL =LFO周波数(0～8)

リターン AL =00H(正常終了)
03H(パラメータエラー)

説　明

FM音源内のLFOの周波数を設定します.

設定されたLFO周波数は, FM音源内の全チャネル(6チャネル)に共通のものとなります. 周波数に0を設定したときは, LFO停止を意味し, 設定する数が大きくなるほど, 周波数が高くなります. 個々のチャネル別のLFOの程度は, 音色データのAMON, AMS/PMSなどによって設定します.

DLの値	0	1	2	3	4	5	6	7	8
周波数(Hz)	解除	3.98	5.56	6.02	6.37	6.88	9.63	48.1	72.2

サウンド	80H
PCMメモリ転送	**機能コード20H**

エントリ AH =20H
DS:ESI =転送元アドレス
EBX =転送先アドレス(0000H～FFFFH)
ECX =転送バイト数

リターン AL =00H(正常終了)
03H(パラメータエラー)
05H(波形メモリ不足)

説　明

ユーザーメモリ内のPCM音声データを, 波形メモリに転送します.

転送元アドレスで指定されるユーザーメモリ上の領域から, 転送先アドレスで指定される波形メモリ上の領域へ, 転送バイト数で指定したバイト数だけのデータが転送されます.

ハードウェアでは, 波形メモリのアクセスは, C2200000H～C2200FFFHのアドレスの4KBのメモリに対して行いますが, BIOSでは, 0000HからFFFFHまでの値で指定できるようになっています.

このオペレーションでは, サウンドデータのヘッダの32バイトをBIOSの格

納領域に転送することはしないので，通常，PCM音源の発音を目的としてサウンドデータをもとに波形データの波形メモリ転送をする場合には，「サウンドの登録(機能コード22H)」を使用してください．

サウンド　80H

音声モードチャネルの設定　機能コード21H

エントリ　AH　＝21H
BL　＝使用チャネル数(0〜8)

リターン　AL　＝00H(正常終了)
04H(波形メモリオーバーフロー)

説　明　音声モードに使用するチャネル数を割り当てます．

BLに指定した数だけ，PCMの下方のチャネル(71〜)から音声モードに割り当てられます．初期値は0です．

音声モードでは，1チャネルにつき8KBのメモリを使用します(4KBずつをダブルバッファとして使う)．このため使用するチャネル数×8KBのメモリを必要とするので，すでに波形メモリ内に楽器モードのサウンドデータが登録されていてこの領域を確保できないときはオーバーフローエラーとなります．オーバーフローを起こした場合には，「サウンドの削除(機能コード23H)」を使用してサウンドデータを削除し，再度このオペレーションを実行します．

音声モードは論理的に8チャネルまでサポートされていますが，4〜5チャネル以上を同時に鳴らすのは処理スピードの限界を超えるため，実質的には困難な動作です．

設定数	使用可能なチャネル
0	なし
1	71
2	71，70
3	71，70，69
4	71，70，69，68
5	71，70，69，68，67
6	71，70，69，68，67，66
7	71，70，69，68，67，66，65
8	71，70，69，68，67，66，65，64

サウンド	80H
サウンドの登録	**機能コード22H**

エントリ
AH　　　＝22H
DS：ESI　＝サウンドデータのアドレス

リターン
AL　　　＝00H(正常終了)
03H(パラメータエラー)
05H(波形メモリ不足)
06H(サウンド内にデータ長が存在しない)

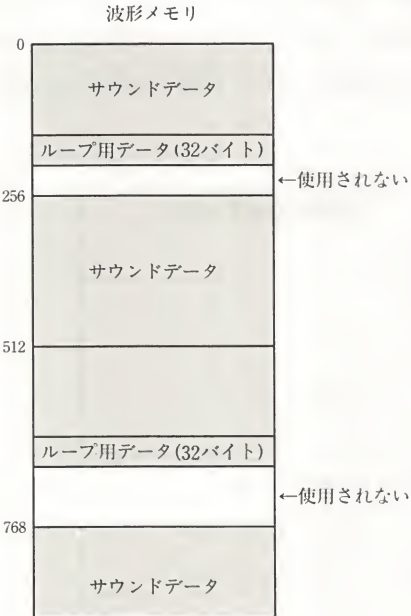
説　明

楽器モードにおいて，サウンドデータ中の波形データを波形メモリに転送し，同時にサウンドデータの32バイトのヘッダ情報をBIOSのワーク領域に登録します．

サウンドデータの後にループ用のデータ(FFH)を32バイト付加します．サウンドデータの登録は256バイトを最小単位として行われるので，256バイト単位で余った波形メモリの部分は使用されません．

(256バイト×N)－32バイトのデータ長のサウンドデータが最も効率が高くなります．

複数のサウンドデータを登録できますが，すでに，波形メモリがいっぱいの場合はエラーとなります．

波形メモリへのサウンドデータの登録の様式を図示します．

サウンド　80H

サウンドの削除　機能コード23H

エントリ	AH	＝23H
	EDX	＝サウンドID
リターン	AL	＝00H(正常終了)
		09H(削除するIDが見つからない)

説　明

楽器モードにおいて，波形メモリ内の波形データを削除し，BIOSの作業領域から，指定されたサウンドIDの情報を削除します．

このとき，波形メモリの空きエリアのガベージコレクションが行われます．オペレーション実行時には，データ移動が起こるためPCM音源の発音は停止します．

なお，サウンドIDにFFFFFFFFHが指定されると，登録されているすべてのサウンドデータが消去されます．

サウンド　80H

PCMサンプリング開始　機能コード24H

エントリ	AH	＝24H
	EDX	＝サンプリング周波数(Hz)
	DS : ESI	＝転送先アドレス
	ECX	＝転送バイト数
	BL	＝トリガレベル(0～127)
リターン	AL	＝00H(正常終了)
		03H(パラメータエラー)
		0CH(サンプリングを強制停止した)

説　明

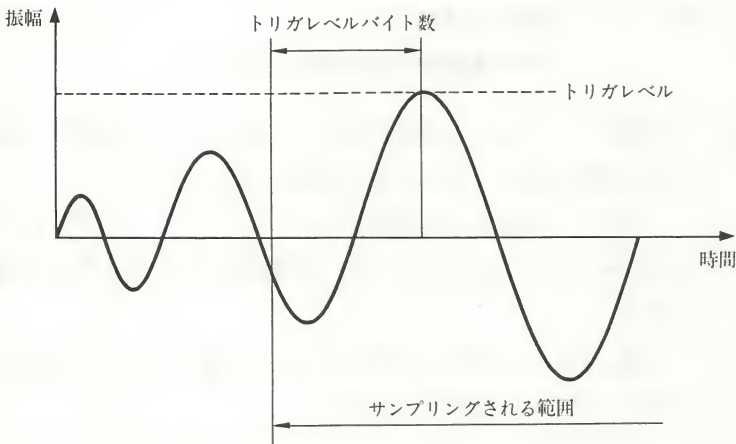
PCMサンプリングを開始します．サンプリング結果は，指定された転送先に格納されます．サンプリング周波数は，Hz単位で指定します．

サンプリングデータが，トリガレベルを超えた時点からサンプリングを開始し，同時に開始時点より前のデータをトリガレベルバイト数分だけサンプリングデータとして加えます．録音ソースは，マイク，CD，LINEなどから選択することができ，ミックスすることも可能です．

なお，サンプリングしたデータを再生させるには，先頭に32バイトのヘッダを付けてサウンドデータを作成しなければなりません．

また，録音時に処理時間のかかる割り込み処理が動作していると，データの取りこぼしが発生し，音質が著しく低下するので注意してください．

転送バイト数は，転送バイト数＝(時間×サンプリング周波数)で計算し，指定します．

トリガレベルとサンプリングの範囲を図解します．

サウンド	80H
音声モードPCM再生	機能コード25H

エントリ

AH　＝25H
BL　＝チャネル番号(64～71)
DH　＝音程(0～127)
DL　＝音量(1～127)
DS:ESI　＝データ領域アドレス

リターン

AL　＝00H(正常終了)
01H(チャネル番号が異常)
02H(キーON中)
03H(パラメータエラー)
06H(サウンド内にデータ長が存在しない)
07H(音声モードが使えないチャネル)
0DH(サウンドデータのヘッダが異常)

説　明

音声モードの再生を行います．

データは，音声モード形式で格納されていなければなりません．サンプリング周波数と音程を計算した結果が20KHzを超える場合は，20KHzで再生されます．

チャネル番号には，PCM音源のチャネルのうち，「音声モードチャネルの設定(機能コード21H)」で音声モードに設定したチャネルの番号を指定します．

DS:ESIには，サウンドデータを格納しておきます．

サウンド	80H
PCMサンプリング中断	機能コード26H

エントリ　AH　＝26H

リターン　AL　＝00H(正常終了)

説　明　PCMサンプリングを開始してから，サンプリングが終了するまでの間に動作を中断させたい場合に使用します．

サウンド	80H
音声モードPCM再生中断	機能コード27H

エントリ　AH　＝27H
　　　　　BL　＝チャネル番号(64～71)

リターン　AL　＝00H(正常終了)
　　　　　　　　03H(パラメータエラー)

説　明　音声モードによるPCM再生を中途で強制的に終了させます．

サウンド	80H
音声モードPCM再生状態参照	機能コード28H

エントリ　AH　＝28H
　　　　　BL　＝チャネル番号(64～71)

リターン　DL　＝演奏状態(0：停止中，0以外：演奏中)
　　　　　AL　＝00H(正常終了)
　　　　　　　　01H(チャネル番号が異常)

説　明　各チャネルの音声モードによる演奏状態をDLに返します．

サウンド	80H
PCM 音源の強制停止	機能コード29H

エントリ AH =29H

リターン AL =00H（正常終了）

説 明 8チャネルすべてをモードに関係なく強制的に停止します。

サウンド	80H
PCM メモリ→メインメモリ転送	機能コード2AH

エントリ

AH	=2AH
EBX	=転送元アドレス（0000H～FFFFH）
DS：ESI	=転送先アドレス
ECX	=転送バイト数

リターン

AL	=00H（正常終了）
	03H（パラメータエラー）
	05H（波形メモリ不足）

説 明

波形メモリ上のPCMデータをメインメモリに転送します。

この機能の使用中は，PCMの発音が停止されます。

また，この機能はBIOSの管理外で動作するので，使用する際には十分に注意してください。

サウンド　80H

PCMメモリ→PCMメモリ転送　機能コード2BH

エントリ	AH	＝2BH
	ESI	＝転送元アドレス(0000H～FFFFH)
	EBX	＝転送先アドレス(0000H～FFFFH)
	ECX	＝転送バイト数
リターン	AL	＝00H(正常終了)
		03H(パラメータエラー)
		05H(波形メモリ不足)

説　明　波形メモリ上のPCMデータを波形メモリに転送します．転送は256バイト単位で行われます．

この機能の使用中は，PCMの発音が停止されます．また，この機能はBIOSの管理外で動作するので，使用する際には十分に注意してください．

サウンド　80H

PCMメモリ転送2　機能コード2CH

エントリ	AH	＝2CH
	DS：ESI	＝転送元アドレス
	EBX	＝転送先アドレス(0000H～FFFFH)
	ECX	＝転送バイト数
リターン	AL	＝00H(正常終了)
		03H(パラメータエラー)
		05H(波形メモリ不足)

説　明　メインメモリ上のPCMデータを波形メモリに転送します．「PCMメモリ転送(機能コード20H)」との相違点は，ループマーク用データ転送防止処理(0FFH→0FEHにデータを変換)が入っていない点です．また，この機能はBIOSの管理外で動作するので，使用する際には十分に注意してください．

サウンド	80H
高品位音声モードPCM再生	機能コード2EH

エントリ　AH　=2EH
BL　=チャネル番号(64〜71)
DH　=音程(0〜127)
DL　=音量(0〜127)
DS：ESI　=データ領域先頭アドレス

リターン　AL　=00H(正常終了)
01H(チャネル番号が異常)
02H(キーON中)
03H(パラメータエラー)
06H(サウンド内にデータ長が存在しない)
07H(音声モードが使えないチャネル)
0DH(サウンドデータのヘッダが異常)

説明　音声モードによる音声出力を開始します．その際，音程，音量の指定が可能です．サンプリング周波数と音程を計算した結果が20KHzを超える場合は，20KHzで再生されます．音声を出力するためには，PCMの割り込みが起こるたびに，「音声モード割り込みエントリ(機能コード51H)」を呼ぶ必要があります．

サウンド	80H
FM音源のみの初期化	機能コード30H

エントリ　AH　=30H

リターン　AL　=00H(正常終了)

説明　TOWNSマウスやTOWNSパッドを動作させる場合には，FM音源LSIにアクセスして，タイマ割り込み機能を使用する必要があります．このオペレーションは，そのような場合にFM音源の初期化を行うためのもので，「ドライバの初期化(機能コード00H)」と違って16KBのワークを必要としません．

このオペレーションはワークが不要です．

サウンド	80H
FM音源レジスタの書き込み	**機能コード31H**

エントリ	AH	＝31H
	BH	＝バンク番号(0，1)
	DH	＝レジスタ番号
	DL	＝データ
リターン	AL	＝00H(正常終了)

説　明

タイマ割り込み設定の際にFM音源のレジスタに直接書き込む際に使用します.

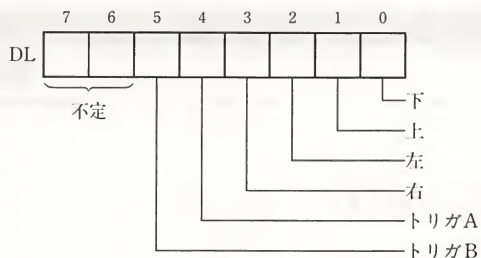
ステータス(割り込みフラグ)の読み出しは,「FM音源ステータス(機能コード10H)」を使用してください.

このオペレーションはワークが不要です.

サウンド	80H
パッド入力1	**機能コード40H**

エントリ	AH	＝40H
	DH	＝ポート番号(0，1)
リターン	AL	＝00H(正常終了)
	DL	＝データ

説　明

パッドポートの値(各ボタンの押下の有無)を参照します.

DLの形式を示します.

各ビットは対応するスイッチがONのときに値が0となります.

RUNボタンが押されると，ビット3とビット2が0となり，SELECTボタンが押されたときはビット1とビット0が，0となります.

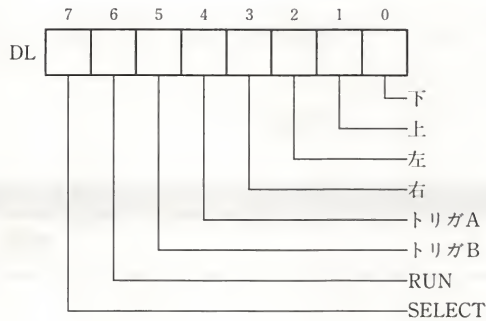
このオペレーションはワークが不要です.

サウンド 80H

パッド入力2 機能コード41H

エントリ　AH　=41H
　　　　　DH　=ポート番号(0，1)

リターン　AL　=00H(正常終了)
　　　　　DL　=データ

説　　明　パッドポートの値(各ボタンの押下の有無)を参照します．
DLの形式を示します．

「パッド入力1(機能コード40H)」との違いは，ビット6がRUNボタンに，ビット7がSELECTボタンに，対応している点です．

十字方向パッドの誤動作を防止するための変換を行います．

このオペレーションはワークが不要です．

サウンド 80H

パッド出力 機能コード42H

エントリ　AH　=42H
　　　　　BL　=書き込みデータ

リターン　AL　=00H(正常終了)
　　　　　DH　=ポート0のデータ
　　　　　DL　=ポート1のデータ

説　明　　パッドポートに値を出力します．リターンは，それぞれのポートの値がDH，DLレジスタに入ります．ポートの値の形式はパッド入力のものと同一です．

このオペレーションはワークが不要です．

サウンド　80H

電子ボリューム設定　機能コード43H

エントリ　AH　＝43H

BL　＝ボリューム番号（0～3）

DH　＝左音量（0～127）

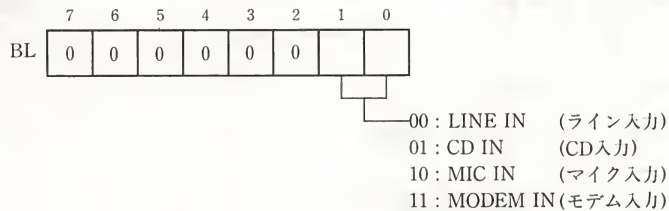
DL　＝右音量（0～127）

リターン　AL　＝00H（正常終了）

説　明　　電子ボリュームの音量を設定します．

音量を設定すると，指定したボリューム番号のミュートが解除されます．

ボリューム番号の値は，入力デバイスを表します．

BLの形式を示します．

このオペレーションはワークが不要です．

サウンド　80H

電子ボリューム初期化　機能コード44H

エントリ　AH　＝44H

リターン　AL　＝00H（正常終了）

説　明　　電子ボリュームを初期化します．

このオペレーション実行後はミュート状態となっており，音が出なくなります．

このオペレーションは，ワークが不要です．

サウンド	80H
電子ボリューム設定読み出し	**機能コード45H**

エントリ	AH	＝45H
	BL	＝ボリューム番号(0～3)
リターン	AL	＝00H(正常終了)
	DH	＝左音量(0～127)
	DL	＝右音量(0～127)

説　明

電子ボリュームの設定状態をDH，DLレジスタに返します．

設定状態を読み込むと，指定したボリューム番号のミュートが解除されます．

ボリューム番号の示す入力デバイスは，「電子ボリューム設定(機能コード43H)」と同じです．

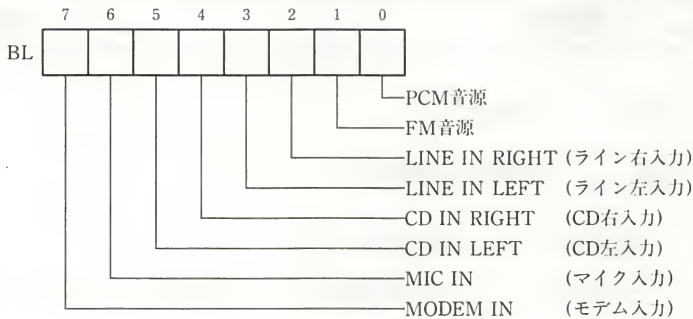
このオペレーションはワークが不要です．

サウンド	80H
電子ボリュームミュート	**機能コード46H**

エントリ	AH	＝46H
	BL	＝ミュート設定
リターン	AL	＝00H(正常終了)

説　明

各電子ボリュームのミュートを設定します．

BLの形式を示します．

ビットが1のときはそのままの音量を維持し，0のときはミュートします．ただし，ボリュームの設定や読み出しのオペレーションを実行すると，PCM音源とFM音源以外のミュートは自動的に解除されてしまいます．

このオペレーションはワークが不要です．

サウンド　80H

電子ボリューム全ミュート　機能コード49H

エントリ　AH　＝49H

DL　＝スイッチ(00H：ミュート，FFH：ミュート解除)

リターン　AL　＝00H(正常終了)

説　明

すべての電子ボリュームのミュートを設定します．

DLが00Hの場合にはミュートがかかり，FFHの場合はミュート解除になります．

このオペレーションはワークが不要です．

サウンド　80H

エンベロープ割り込みエントリ　機能コード50H

エントリ　AH　＝50H

リターン　なし

説　明

PCM楽器モードで，10ms毎の割り込み処理を行うためのエントリです．

サウンド	80H
音声モード割り込みエントリ	機能コード51H

エントリ　AH　＝51H

リターン　なし

説明　PCM音声モードでの割り込み処理エントリです．

音声モード割り込みが発生したら，このエントリを呼び出すようにしないと，正しく発音できません．

6.6　サウンドBIOSの拡張機能

当初 FMTOWNS に搭載されていた PCM 音源は 8 ビットでしたが，FMTOWNS II MX などから，16 ビットステレオが可能になり，MPC（Multimedia Personal Computer）規格に基づく，新 PCM 音源もサポートされました．この機能は CD や DAT のレベルと同等で，高音質の音楽データを処理することが可能です．新 PCM 音源では，次の属性を組み合わせることができます．

サンプリング周波数	11.025kHz，22.050kHz，44.100kHz
サンプリングビット数	8 ビット，16 ビット
種別	モノラル，ステレオ

このうちサンプリング周波数は MPC の推奨値で，規格上は任意の値がとれますが，互換性を考慮して固定値が選ばれています．

この形式のデータを収容するファイルは，マイクロソフト・ウエーブ・フォーム・データ形式（以下「WAVE フォーマット」という）で，拡張子は “.WAV” です．

6.6.1　リングバッファの働きとオーバーラン，アンダーラン

今回の拡張機能では，長時間の録音／再生に対応するため，リングバッファを使用しています．

リングバッファは，バッファの先端と終端が文字どおりリング状につながっているメモリで，終端まで来るとそのまま先端に接続して運用されます．したがって，その堺目でのとぎれがなく，あたかもリングのように連続してデータを扱うことができます．

しかし，たとえば録音（サンプリング）のとき，リングバッファ状のデータが一周して，アプリケーション側に渡らないうちにオーバーライトしなければならない状態になると，以前のデータは消えてしまいます．そして，もしそのままサンプリングを続けてもデータの連続性は断たれているので，システムはエラー発生とみなして処理を中止します．このような現象を「オーバーラン」といいます．

一方，再生の場合は，アプリケーション側から来たデータがリングバッファに渡されて，それをシステムが音声に変換します．もし，アプリケーションのデータが残っているのに，リングバッファのデータが再生され尽くした場合は，同様に再生の中断が起こります．これが「アンダーラン」です．

6.6.2 リングバッファ管理テーブルとリングバッファの容量

リングバッファのメモリ区域は1個当たり4,096バイトで，最低2個必要ですが，最大個数はメモリが許す限り使用することができます．複数個のリングバッファは，0番から順番に使用され，ひとつのバッファの終端まで来たら次のバッファに移ります．分割されてはいますが，実質的には一連のものとみなすことができ，最後の番号のバッファの終端は0番のバッファの先端に接続されています．すなわち，結果的にひとつのリングを形成していることになります．

そして，これらはリングバッファ管理テーブル（表II-6-11）により管理されています．

表中のリングバッファ総数（n）は，使用するリングバッファの個数を定義します．

アプリケーション用バッファ位置（0～n−1）は，録音時に取り出すデータが入っているリングバッファの番号を指します．また，再生時は，再生データを転送すべきリングバッファの番号を意味します．

システム用処理バッファ位置（0～n−1）は，システムが処理中（録音中または再生中）のリングバッファの番号を表します。

バッファアドレスは，リングバッファの番号に対応した実際の先頭アドレスが入ります．

システムリザーブの部分は，システムが使用している領域で，アプリケーションには開放されていません．

リングバッファ管理テーブルを作成するには，リングバッファ総数のみ定義しておいて，後述の「リングバッファ管理テーブル作成（機能コード6BH）」を使えば，内容値が自動設定されます．

なお，個別のバッファの大きさは4,096バイトですが，全体の容量は

$$4096 \times (\text{リングバッファ総数} + 1)$$

だけ用意します．これは，個別のバッファが確実にページ境界に割り当てられるようにするためです．

▼表 II-6-11　リングバッファ管理テーブルの構造

	サイズ	内　容
0	4	リングバッファ総数（n）
4	4	アプリケーション用処理バッファ位置（0〜n−1）
8	4	システム用処理バッファ位置（0〜n−1）
12	4	0番目バッファアドレス
16	8	システムリザーブ
24	4	1番目バッファアドレス
28	8	システムリザーブ
	≈	≈
	4	n−1番目バッファアドレス
	8	システムリザーブ

6.6.3　8ビットのみのサポート時の制約事項

PCM 音源で 8 ビットのみがサポートされている状態（「録音／再生機能サポート状況の取得（機能コード 67H）」で参照した内容が 16 ビット PCM なしの場合）では，新 PCM 音源が使われず，旧 PCM 音源により動作します．

このため，サウンド BIOS とその拡張機能が重複して作用するケース，例えば再生音量のミュートなどでは，相互に影響し，最後に実行されたほうの結果が残ります．

ハードウェアは新 PCM 音源を使用しないため，録音時はモノラルしかサポートされず，録音時の処理関数も実行できません．また，録音の即時復帰はできず，すべて完了復帰となります．

録音時間は，「録音前準備（機能コード 70H）」で録音データ格納アドレスの指定があったときは，「録音開始（機能コード 71H）」のサンプリングデータ長で指定されたサイズを埋めるまでとなりますが，格納アドレスがないときは，リングバッファの総容量を埋めた時点で打ち切られます．

再生に当たっては，左右に振り分けする方法でのステレオ対応ができます．再生時の各モードで，旧 PCM 音源が持っているチャネルは，次のように占有されます．

8 ビットモノラル	1ch
8 ビットステレオ	2ch
16 ビットモノラル	2ch
16 ビットステレオ	4ch

また，16 ビット WAVE ファイルを再生すると，設定音量に対し実際の音量は 8 段階になります．

6.6.4 サウンド BIOS 拡張機能一覧

サウンド BIOS 拡張機能の一覧を，表 II–6–12 に示します．

▼表 II-6-12　サウンド BIOS 拡張機能一覧

機能名称	機能コード
拡張機能の初期化	60H
拡張機能の終了	61H
録音/再生状態の初期化	63H
再生音量のミュート	64H
再生音量の設定	65H
再生音量の取得	66H
録音/再生機能サポート状況の取得	67H
録音/再生状態の取得	68H
WAVE ファイルの情報の設定	69H
WAVE ファイルの情報の取得	6AH
リングバッファ管理テーブル作成	6BH
リングバッファ管理テーブルおよびリングバッファアドレスの取得/設定	6CH
録音前準備	70H
録音開始	71H
録音強制終了	72H
録音データ格納アドレスの取得	73H
再生前準備	78H
再生開始	79H
再生強制終了	7AH
再生データアドレスの取得	7BH

6.6.5 エラーコード一覧

サウンド BIOS 拡張機能のエラーコードの一覧を，表 II–6–13 に示します．

▼表 II-6-13　エラーコード一覧

コード	意味
0	正常終了
20	動作環境不備
21	パラメータエラー
22	WAVE フォーマットのデータでない
23	情報不足
24	リングバッファ総数が 1 以下
25	リングバッファ領域が適当でない
26	録音強制終了
27	すでに初期化済
28	リンクバッファ管理テーブルが未作成
29	現在録音または再生中

エラーコードはすべて 10 進値です．

6.7　サウンドBIOS拡張機能リファレンス

サウンドBIOSの拡張機能について，個別に説明します．

サウンド	80H
拡張機能の初期化	機能コード60H

エントリ　AH　=60H

リターン　EAX　=0（正常終了時）

説　明　サウンドBIOS拡張部分のハードウェアとソフトウェアの初期化を行うオペレーションで，拡張機能を使用するとき，事前に実行します．あくまで拡張部分のためのものなので，このオペレーション以前に，サウンドBIOSそのものの「ドライバの初期化（機能コード00H）」がなされていなければなりません．

この機能が呼ばれると，次の処理が行われます．

・拡張機能用の作業エリアの初期化
・拡張機能用のハードウェアの初期化
・再生音量を0に設定

サウンド	80H
拡張機能の終了	機能コード61H

エントリ　AH　=61H

リターン　EAX　=0（正常終了時）

説　明　サウンドBIOS拡張部分のハードウェアとソフトウェアの終了処理を行うオペレーションです．拡張機能を使い終えたときに実行します．

もし，プログラムでサウンドBIOS全体の終了処理を行うときは，先にこのオペレーションが実行されていなければなりません．C言語で，サウンドライブラリの終了関数（SND_end）を呼ぶ場合も同じです．

サウンド	80H
録音／再生状態の初期化	機能コード63H

エントリ　AH　=63H

リターン　EAX　=0（正常終了時）

説　明　WAVEデータの録音／再生状態の初期化を行うオペレーションです．WAVEデータの処理からSNDデータの処理（例えば,「PCMサンプリング開始（機能コード24H)」など）に移るとき，事前にこのオペレーションを実行します．

実行後，再生音量が0となります．

サウンド	80H
再生音量のミュート	機能コード64H

エントリ　AH　=64H

AL　=ミュートモード

（0：ミュート解除，1：ミュート設定，2：ミュート状態取得）

リターン　EAX　=0（正常終了時）

BL　=ミュート状態（CL＝2のときのみ有効）

0：ミュート解除中

1：ミュート中

説　明　WAVEデータの再生音量のミュート制御およびミュートの状態を参照するためのオペレーションです．戻り値のBLは，ミュート状態取得のときだけ有効で，その他のときは意味を持ちません．

サウンド	80H
再生音量の設定	**機能コード 65H**

エントリ　AH　=65H
　　　　　BL　=左音量（0〜127）
　　　　　BH　=右音量（0〜127）

リターン　EAX　=0（正常終了時）

説　明　WAVE データの左右の再生音量を設定します．もし，ミュート中の場合でも，この設定が優先され，ミュートが解除されます．

サウンド	80H
再生音量の取得	**機能コード 66H**

エントリ　AH　=66H

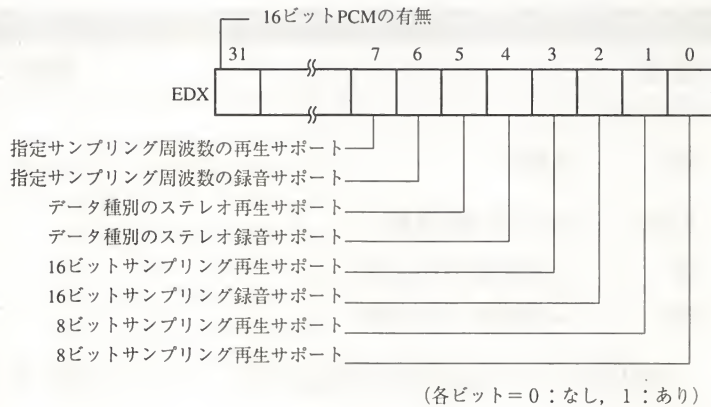
リターン　EAX　=0（正常終了時）
　　　　　BL　=左音量（0〜127）
　　　　　BH　=右音量（0〜127）

説　明　WAVE データの左右の再生音量を参照し，それらの値を BL と BH に収容します．

サウンド	80H
録音／再生機能のサポート状況の取得	機能コード67H

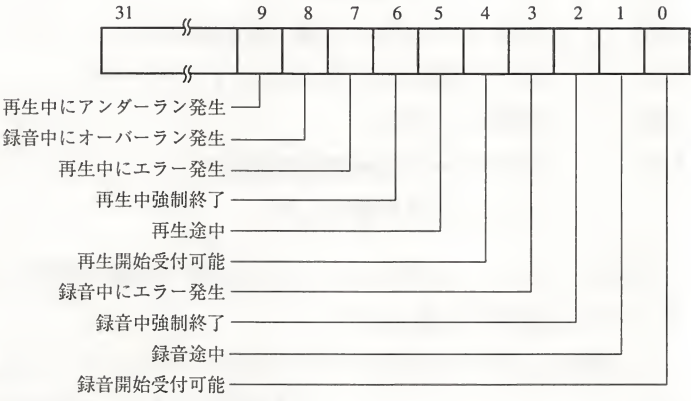
エントリ　AH　=67H
　　　　　EDX　=指定サンプリング周波数

リターン　EAX　=0（正常終了時）
　　　　　EDX　=録音／再生機能サポート状況

説　明　現在の動作環境でサポートされている機能について，指定されたサンプリング周波数における状態を参照し，EDXに収容します．EDXの内容は，次のようになっています．

サウンド　80H
録音／再生状態の取得　機能コード68H

エントリ　AH　＝68H

リターン　EAX　＝0（正常終了時）
EDX　＝録音／再生状態

説　明　現在の録音／再生の状態を参照し，EDXに収容します．EDXの内容は，次のようになっています．

オーバーランとアンダーランについては，6.6.1の説明を参照してください．

サウンド　80H
WAVEファイルの情報の設定　機能コード69H

エントリ　AH　＝69H
ES：EDI　＝WAVEファイルの情報を格納するバッファアドレス
EDX　＝サンプリング周波数（Hz）
CH　＝サンプリングビット数（8または16）
CL　＝データ種別（1：モノラル，2：ステレオ）
EBX　＝PCMデータ長

リターン　EAX　＝0（正常終了時）
EDX　＝PCMデータ格納開始相対位置

説　明　WAVEファイルの先頭には，44バイトのファイル情報の領域が必要です．そして，その領域の内容値は，ここで述べるオペレーションで設定できます．

サウンド	80H
WAVE ファイルの情報の取得	機能コード 6AH

エントリ

AH　=6AH
ES：EDI　=WAVE ファイルの情報が格納されているバッファアドレス
ECX　=バッファサイズ（通常 44）

リターン

EAX　=0（正常終了時）
　23（エラー：ファイル情報サイズがバッファサイズより大きい）
EDX　=サンプリング周波数（Hz）
CH　=サンプリングビット数（8 または 16）
CL　=データ種別（1：モノラル，2：ステレオ）
EBX　=PCM データ長
ESI　=PCM データ格納開始相対位置
　（=ファイル情報サイズ）

説明

WAVE ファイルの先頭にあるファイル情報の領域を参照し，EDX などのレジスタに主要項目を転送します．

この領域の大きさは通常 44 バイトですが，WAVE ファイルを作成したアプリケーションによっては，もっと大きな場合があり，その場合はエラーが発生します．そのような WAVE ファイルを読むときは，バッファサイズを大きくとらなければなりません．

または，あらかじめバッファサイズを大きめにとっておき，このオペレーションを実行して，ESI の値を実際のファイル情報サイズとして利用する方法もあります．この場合，その領域に続いて PCM データが格納されているので，ESI の値は同時に格納開始相対アドレスをも意味しています．

サウンド　80H

リングバッファ管理テーブル作成　機能コード 6BH

エントリ　AH　=6BH

DS：ESI　=リングバッファ領域アドレス

ES：EDI　=リングバッファ管理テーブルアドレス

リターン　EAX　=0（正常終了時）

説　明　WAVE データの録音／再生に使われるリングバッファと，それを管理するテーブルとを連結するためのオペレーションです．管理テーブルの先頭にはリングバッファの総数を入れておき，このオペレーションを実行すると，リングバッファを 4KB（4,096 バイト）単位で区切り，個別の区域のアドレスをテーブル内のバッファアドレスに転送して関係づけが行われます．

リングバッファ管理テーブルは，リングバッファ区域数を n とすると，

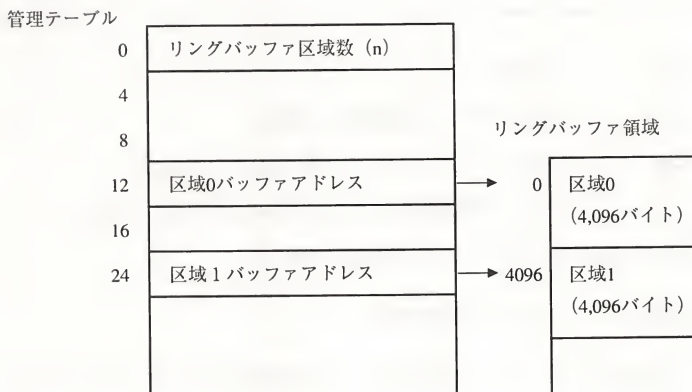
$$12 + 12 \times n \text{〔バイト〕}$$

だけの大きさが必要です．

また，リングバッファの大きさは，すべてを確実にページ境界に割り付けするため，区域 0 を境界にシフトする目的のダミー 1 個分を加えて

$$4,096 \times (n + 1) \text{〔バイト〕}$$

だけ用意します．ここでいう“ページ”は，アドレス下位が 000H で始まる 4,096 単位の領域です．

以上の関係を図示すると，次のようになります．

サウンド	80H
リングバッファ管理テーブルおよびリングバッファアドレスの取得／設定	機能コード 6CH

エントリ

AH	＝6CH
BL	＝処理モード（0：取得，1：設定）
（以下，BL＝1の場合のみ有効）	
DS：ESI	＝リングバッファ領域アドレス
ES：EDI	＝リングバッファ管理テーブルアドレス

リターン

EAX	＝0（正常終了時）
（以下，BL＝0の場合のみ有効）	
DS：ESI	＝リングバッファ領域アドレス
ES：EDI	＝リングバッファ管理テーブルアドレス

説　明

リングバッファと，リングバッファ管理テーブルのアドレスを設定（BIOSに知らせる），または既設定値を参照するオペレーションです．

ただし，録音／再生実行中は，リングバッファが使用中のため，このオペレーションを実行するとエラーになります．

参照したアドレスはBIOS内部で管理されているものであり，場合によっては設定値と異なることがあるので注意が必要です．

サウンド	80H
録音前準備	機能コード 70H

エントリ

AH	＝70H
EDX	＝サンプリング周波数（Hz）
CH	＝サンプリングビット数（8または16）
CL	＝データ種別（1：モノラル，2：ステレオ）
ES：EDI	＝パラメータの格納アドレス

リターン

EAX	＝0（正常終了時）

説　明

録音に先立って，WAVEデータの情報を設定するオペレーションです．

パラメータにより，録音データの格納アドレスを設定した場合，システムが

リングバッファを参照して書き込みます．

また，録音処理と同期して呼び出される関数（サブルーチン）の指定があるときは，一定間隔でそれが実行されます．ただし，関数の処理時間が長いと，録音が途切れたり，停止することがあるので注意が必要です．

〔パラメータ〕

(ES：EDI)

0	DW	録音データ格納アドレスのオフセット*1
4	DW	録音データ格納アドレスのセレクタ
8	DW	録音処理と同期して行う関数のオフセット*1
12	DW	録音処理と同期して行う関数のセレクタ
16	DW	処理関数用ローカルスタックのオフセット*2
20	DW	処理関数用ローカルスタックのセレクタ
24	DW	処理関数実行時に設定される DS
28	DW	処理関数実行時に設定される ES
32	DW	処理関数実行時に設定される FS
36	DW	処理関数実行時に設定される GS

*1：該当項目を使用しないときは，0 を設定．
*2：処理関数を使用するときは，スタック領域の底のオフセットを設定．

〔処理関数の呼び出し間隔例〕

サンプリング周波数	8 ビット		16 ビット	
	モノラル	ステレオ	モノラル	ステレオ
11.025kHz	約 371ms	約 185ms	約 185ms	約 92ms
22.050kHz	約 185ms	約 92ms	約 92ms	約 46ms
44.100kHz	約 92ms	約 46ms	約 46ms	約 23ms

サウンド	80H
録音開始	機能コード 71H

エントリ

AH　=71H

BH　=録音モード（1：指定されたデータ長まで録音（完了復帰），
2：強制終了（72H）で終了（即時復帰））

BL　=トリガレベル（0〜127）

ECX　=サンプリングデータ長

リターン

EAX　=0（正常終了時）

説　明

WAVE データの録音開始オペレーションです．

完了復帰の場合，録音位置がサンプリングデータ長に達した段階で終了します．このため，このオペレーションを開始すると，完了するまでオペレーションの処理を継続します．

強制終了で終了させる場合は，録音は「録音強制終了（機能コード 72H）」を実行した時点で終了します．したがって，72H を実行するためにはこのオペレーションを抜けていなければならないので，「録音開始（機能コード 71H）」とともに，ただちに復帰します（即時復帰）．

録音ソースは，サウンド BIOS を使用して選択でき，マイク，CD，LINE-IN などが使えます．ミュートを解除して電子ボリュームを利用し，ミキシングした音をサンプルすることもできます．

注意しなければならないのは，再生音量が設定されていると信号の回り込みが起き，ハウリングの原因となるので，事前に再生音量をゼロにするか，ミュートするかのいずれかに設定することが必要です．

録音は，サンプリングデータの絶対値が指定トリガレベルを超えた時点から開始されます．トリガレベルの設定は 8 ビットなので，16 ビットサンプリングの場合は，サンプリングデータ絶対値の上位 8 ビットと比較が行われます。

録音動作に入ると，PCM データをリングバッファに格納し，同時にシステムのバッファ位置カウンタを更新します。完了復帰の場合は，この値がサンプリングデータ長と等しくなると終了します．また，「録音前準備（機能コード 70H）」で録音データのアドレスを指定した場合は，アプリケーション用処理バッファ位置の更新もシステムが行います．

サウンド	80H
録音強制終了	機能コード72H

エントリ　AH　=72H

リターン　EAX　=0（正常終了時）

説　明

WAVEデータの録音を終了させるオペレーションです．基本的には「録音開始（機能コード71H)」オペレーションで即時復帰を指定した場合の終了動作のために使われます．

完了復帰の場合は，録音終了まで「録音開始（機能コード71H)」が続行するので，そのままでは強制終了させることができません．どうしても強制終了させたいときは，割り込み処理の中でこのオペレーションを実行する方法があります．

サウンド	80H
録音データ格納アドレスの取得	機能コード73H

エントリ　AH　=73H

リターン　EAX　=0（正常終了時）
ES：EDI　=録音データアドレス

説　明

「録音前準備（機能コード70H)」で録音データ格納アドレスを指定した場合，このオペレーションでES（セレクタ）とEDI（オフセット）に録音データアドレスを取得することができます．

その他の場合も，このオペレーションは一応正常終了しますが，録音データアドレスの値は無意味です．

サウンド	80H
再生前準備	機能コード78H

エントリ　AH　=78H
EDX　=サンプリング周波数（Hz）
CH　=サンプリングビット数（8または16）
CL　=データ種別（1：モノラル，2：ステレオ）
ES：EDI　=パラメータの格納アドレス

リターン　EAX　=0（正常終了時）

説　明　再生に先立って，WAVE データの情報を設定するオペレーションです．

パラメータにより，再生データの先頭アドレスを設定した場合，システムがリングバッファを参照して読み出します．

また，再生処理と同期して呼び出される関数（サブルーチン）の指定があるときは，一定間隔でそれが実行されます．ただし，関数の処理時間が長いと，再生が途切れたり，停止したりすることがあるので注意が必要です．

〔パラメータ〕

(ES : EDI)

オフセット	型	内容
0	DW	再生データ先頭アドレスのオフセット*1
4	DW	再生データ先頭アドレスのセレクタ
8	DW	再生処理と同期して行う関数のオフセット*1
12	DW	再生処理と同期して行う関数のセレクタ
16	DW	処理関数用ローカルスタックのオフセット*2
20	DW	処理関数用ローカルスタックのセレクタ
24	DW	処理関数実行時に設定される DS
28	DW	処理関数実行時に設定される ES
32	DW	処理関数実行時に設定される FS
36	DW	処理関数実行時に設定される GS

*1：該当項目を使用しないときは，0 を設定．
*2：処理関数を使用するときは，スタック領域の底のオフセットを設定．

〔処理関数の呼び出し間隔例〕

サンプリング周波数	8 ビット		16 ビット	
	モノラル	ステレオ	モノラル	ステレオ
11.025kHz	約 371ms	約 185ms	約 185ms	約 92ms
22.050kHz	約 185ms	約 92ms	約 92ms	約 46ms
44.100kHz	約 92ms	約 46ms	約 46ms	約 23ms

サウンド	80H
再生開始	機能コード79H

エントリ　AH　=79H
ECX　=サンプリングデータ長

リターン　EAX　=0（正常終了時）

説　明

WAVEデータの再生開始オペレーションです．

録音の場合は復帰タイミングが2種類ありますが，再生に関しては即時復帰のみで，再生開始と同時にこのオペレーションから戻ります．そして，あとはユーザープログラムと並行して再生処理が続行します．

再生動作に入ると，リングバッファに収容されているPCMデータを読み出し，同時にシステムのバッファ位置カウンタを更新します．そして，この値がサンプリングデータ長と等しくなると再生を終了します．また，「再生前準備（機能コード78H）」で再生データのアドレスを指定した場合は，アプリケーション用処理バッファ位置の更新もシステムが行います．

サウンド	80H
再生強制終了	機能コード7AH

エントリ　AH　=7AH

リターン　EAX　=0（正常終了時）

説　明

WAVEデータの再生を強制終了させるオペレーションです．

完了復帰の場合は，録音終了まで「録音開始（機能コード71H）」オペレーションが続行するので，そのままでは強制終了させることができません．どうしても強制終了させたいときは，割り込み処理の中でこのオペレーションを実行する方法があります．

サウンド　80H

再生データアドレスの取得　機能コード7BH

エントリ　AH　=7BH

リターン　EAX　=0（正常終了時）

ES：EDI　=再生データアドレス

説　明　「再生前準備（機能コード78H)」で再生データアドレスを指定した場合，このオペレーションでES（セレクタ）とEDI（オフセット）に再生データアドレスを取得することができます．

その他の場合も，このオペレーションは一応正常終了しますが，録音データアドレスの値は無意味です．

第 7 章

CD-ROM BIOS

FMTOWNS の CD-ROM BIOS は，FMR シリーズ用に開発された BIOS にFMTOWNS 独自の機能を付け加えたものです．

この章では，この CD-ROM BIOS について解説します．

7.1 CD-ROM BIOS 一覧

CD-ROM BIOS は FMR シリーズと共通のリアル BIOS であり，ディスク BIOS と同じ INT 93H で呼び出します．

なお，CD と CD-ROM はハード的には同一のメディアであり，ドライブ装置も同一です．したがって以下の説明では，CD と CD-ROM の総称として〝CD〞を使用し，ドライブ装置を CD ドライブと呼ぶことにします．

CD-ROM BIOS は次のように 5 種類に分類できます．

1．ドライブ状態の設定／参照

CD のセクタ長(2048/2336/2340の 3 種類)を調べたり，CD ドライブの設定をセクタ長に合わせます．

2．シーク

CD の最内周(トラック 0)への移動であるリストア，論理セクタ番号および，時間指定による任意の位置へのシークを行います．

3．データの読み取り

論理セクタ，または時間指定でデータを読み取り，メモリに転送します．1MB を越えるアドレスへデータ転送を行う場合には，拡張オペレーションを使用します．拡張オペレーションでは，0～4GB の範囲が転送可能です．

4．演奏

演奏に関係するものとして，音楽演奏スタート，一時停止，一時停止解除，音楽演奏ストップ，リピート動作などがあります．

5．演奏情報読み出し

CDの詳細な情報や演奏状態などを知ることができます．

表II-7-1に，CD-ROM BIOS一覧を示します．

▼表II-7-1　CD-ROM BIOS一覧

機能名称	機能コード
ドライブモードの設定	00H
ドライブモードの読み取り	01H
ドライブステータス情報の読み取り	02H
シリンダ0へのシーク	03H
指定位置へのシーク(論理セクタ指定)	04H
データの読み取り(論理セクタ指定)	05H
データの読み取り(論理セクタ指定)〈拡張〉	05H
指定位置へのシーク(時間指定)	14H
データの読み取り(時間指定)	15H
データの読み取り(時間指定)〈拡張〉	15H
音楽演奏スタート	50H
音楽演奏スタート(リピート機能)〈拡張〉	50H
音楽演奏スタート(回数指定のあるリピート機能)〈拡張〉	50H
音楽演奏情報の読み取り	51H
音楽演奏ストップ	52H
CDドライブ停止時間の設定〈拡張〉	52H
音楽演奏状態の読み取り	53H
CD情報の読み取り	54H
音楽演奏一時停止	55H
音楽演奏一時停止解除	56H

7.2　CD-ROM BIOSオペレーションの共通事項

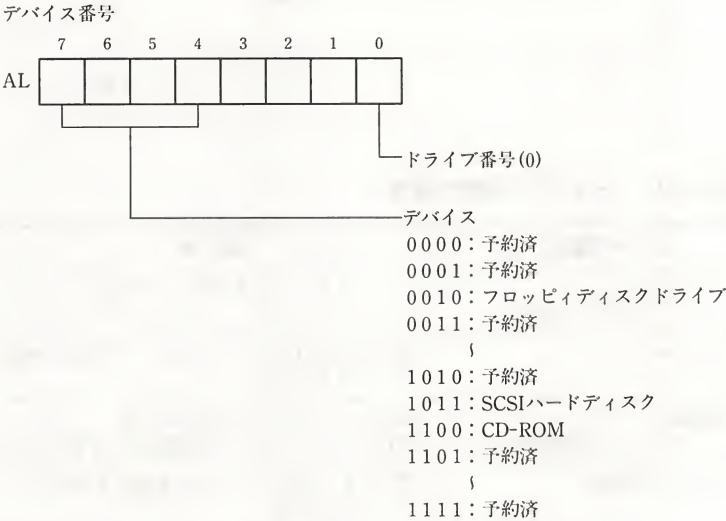
ここでは，CD-ROM BIOSの各オペレーションに共通する事項について解説します．

●デバイス番号

CD-ROM BIOSは，各種のデバイスに対応するように作られており，BIOSをコールする場合には，ALレジスタにデバイス番号を指定します．

図II-7-1にALの形式を示します．

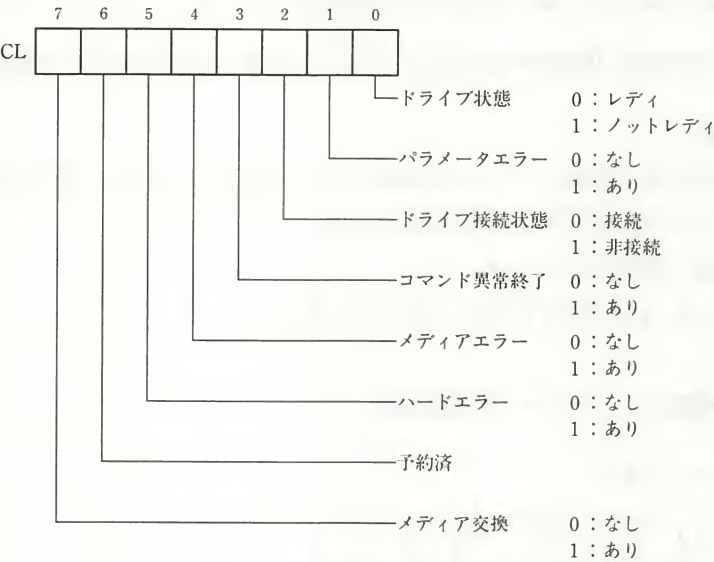
この図のように，CD-ROM BIOSではC0Hを指定します．

▼図II-7-1　デバイス番号の形式

●ハードエラー情報

CD-ROM BIOSのすべてのオペレーションで，ハードエラーが発生する可能性があります．ハードエラーが発生すると，リターンのAHが80Hとなり，CXにその詳細情報が返されます．

図II-7-2にCX(CL)の形式を示します．各エラーの意味は表II-7-2に示します．

なお，エラー発生時，BIOSではリトライを行いません．

▼図II-7-2　ハードエラー情報の形式

▼表II-7-2　ハードエラー情報の意味

エラーの種類	意　味
ノットレディ（ドライブ状態）	コンパクトディスクが入っていない．
パラメータエラー	指定されたコマンド，パラメータに誤りがある．プログラムミス．
非接続（ドライブ接続状態）	指定したドライブが存在しない． CDドライブユニットの電源が入っていない．
コマンド異常終了	CDドライブがコマンドを異常終了させた． リトライが必要．
メディアエラー	メディアがキズなどによりアクセスできない． オーディオトラックをROMリードした． ROMトラックをオーディオアクセスした．
ハードエラー	コマンド実行中，ハード的にエラーが発生し，アクセスが中断された． 自己診断異常．
メディア交換	CDが交換された．

7.3　CD-ROM BIOS リファレンス

CD-ROM BIOS について個別に詳しく解説します。

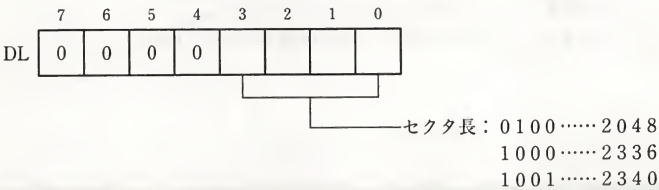
CD-ROM	INT 93H
ドライブモードの設定	機能コード00H

エントリ
AH　=00H
AL　=デバイス番号(C0H)
CH　=00H
DL　=ドライブモード

リターン
AH　=00H(正常終了)
　02H(デバイス番号エラー)
　80H(ハードエラー)
CX　=エラー情報(AH が80Hの場合)

説　明
CD のセクタ長を設定します.

システム起動時にセクタ長は2048バイトに設定されていますが，それを変更する場合に使用します．DL にセクタ長を指定します.

DL の形式を示します.

セクタ長
2048：ユーザーデータ領域のみ(物理セクタ長と同じ)
2336：ユーザーデータ領域+ECC／EDC
2340：ヘッダ+ユーザーデータ領域+ECC／EDC

CD-ROM	INT 93H
ドライブモードの読み取り	機能コード01H

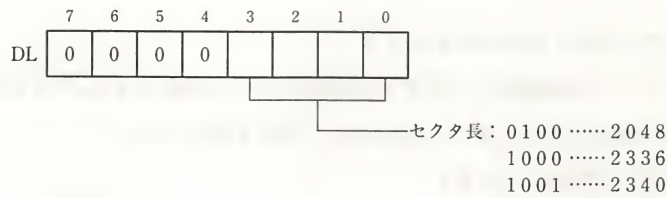
エントリ　AH　＝01H
AL　＝デバイス番号(C0H)
CH　＝00H

リターン　AH　＝00H(正常終了)
02H(デバイス番号エラー)
80H(ハードエラー)
DL　＝ドライブモード
CX　＝エラー情報(AHが80Hの場合)

説明

CDのセクタ長を読み出します.

DLに「ドライブモードの設定(機能コード00H)」で設定したセクタ長が返されます.

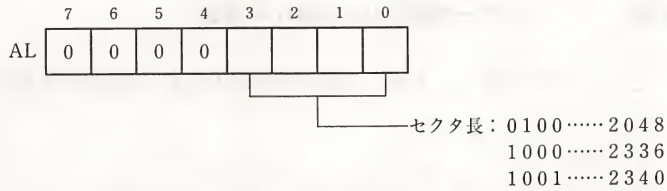
DLの形式を示します.

セクタ長
2048：ユーザーデータ領域のみ(物理セクタ長と同じ)
2336：ユーザーデータ領域+ECC／EDC
2340：ヘッダ+ユーザーデータ領域+ECC／EDC

CD-ROM	INT 93H
ドライブステータス情報の読み取り	機能コード02H

エントリ　AH　＝02H
AL　＝デバイス番号(C0H)
CH　＝00H

リターン	AH	=00H(正常終了)
		02H(デバイス番号エラー)
		80H(ハードエラー)
	AL	=ドライブモード
	BH	=00H
	BL	=最大論理セクタ数(上位バイト)
	DX	=最大論理セクタ数(下位バイト)
	CX	=エラー情報(AHが80Hの場合)

説　明

現在セットされているCDのセクタ数とセクタ長を得ます．
ALにセクタ長が返されます．
ALの形式を示します．

BLとDXに返される最大論理セクタ数は，ユーザーデータ領域を示しているため，セクタ長にかかわらず一定です．

CD-ROM　INT 93H
シリンダ0へのシーク　機能コード03H

エントリ	AH	=03H
	AL	=デバイス番号(C0H)
	CH	=00H

リターン	AH	=00H(正常終了)
		02H(デバイス番号エラー)
		80H(ハードエラー)
	CX	=エラー情報(AHが80Hの場合)

説　明

ヘッドをCDの先頭位置(最内周)へ移動(シーク)します．

CD-ROM　INT 93H
指定位置へのシーク（論理セクタ指定）　機能コード04H

エントリ	AH	＝04H
	AL	＝デバイス番号（C0H）
	CH	＝00H
	CL	＝論理セクタ番号（上位バイト）
	DX	＝論理セクタ番号（下位バイト）
リターン	AH	＝00H（正常終了）
		02H（デバイス番号エラー）
		10H（音楽演奏中）
		80H（ハードエラー）
	CX	＝エラー情報（AHが80Hの場合）

説　明　CDドライブのヘッドを，任意の論理セクタへ移動させます．

CD-ROM　INT 93H
データの読み取り（論理セクタ指定）　機能コード05H

エントリ	AH	＝05H
	AL	＝デバイス番号（C0H）
	CH	＝00H
	CL	＝論理セクタ番号（上位バイト）
	DX	＝論理セクタ番号（下位バイト）
	BX	＝読み取りするセクタ数
	DS：DI	＝バッファのアドレス
リターン	AH	＝00H（正常終了）
		02H（デバイス番号エラー）
		10H（音楽演奏中）
		80H（ハードエラー）
	BX	＝残りのセクタ数
	CX	＝エラー情報（AHが80Hの場合）

説　明　CL，DX で指定した論理セクタ位置から，BX に指定されたセクタ数だけのデータを読み出し，指定したバッファ(DS：DI)に格納します．

また，読み取り時にエラーが発生した場合には，残りのセクタ数が BX に返されます．

CD-ROM	INT 93H
データの読み取り(論理セクタ指定)〈拡張〉	機能コード05H

エントリ

AH	＝05H
AL	＝デバイス番号(C0H)
CH	＝FFH
CL	＝論理セクタ番号(上位バイト)
DX	＝論理セクタ番号(下位バイト)
BX	＝読み取りするセクタ数
SI	＝バッファアドレス(上位2バイト)
DI	＝バッファアドレス(下位2バイト)

リターン

AH	＝00H(正常終了)
	02H(デバイス番号エラー)
	10H(音楽演奏中)
	80H(ハードエラー)
BX	＝残りのセクタ数
CX	＝エラー情報(AH が80H の場合)

説　明　「データの読み取り(論理セクタ指定)(機能コード05H，CH＝00H)」では，アドレス空間は最大 1MB までに限定されますが，この拡張オペレーションでは，32ビットのアドレス空間に対応し，4GB までのメモリをバッファとして使用できます．CH に指定する値が FF となり，SI と DI でバッファアドレスを物理アドレスで指定する点以外は，「データの読み取り(論理セクタ指定)(機能コード 05H，CH＝00H)」と同様です．

CD-ROM	INT 93H
指定位置へのシーク(時間指定)	機能コード14H

エントリ	AH	＝14H
	AL	＝デバイス番号(C0H)
	CH	＝00H
	CL	＝分
	DH	＝秒
	DL	＝フレーム
リターン	AH	＝00H(正常終了) 02H(デバイス番号エラー) 10H(音楽演奏中) 80H(ハードエラー)
	CX	＝エラー情報(AHが80Hの場合)

説　明　CDドライブのヘッドを，任意の時間位置(分，秒，フレーム)に移動します．

CD-ROM	INT 93H
データの読み取り(時間指定)	機能コード15H

エントリ	AH	＝15H
	AL	＝デバイス番号(C0H)
	CH	＝00H
	CL	＝分
	DH	＝秒
	DL	＝フレーム
	BX	＝読み取りするセクタ数
	DS：DI	＝バッファのアドレス

リターン　AH　=00H(正常終了)
　　　　　　　　02H(デバイス番号エラー)
　　　　　　　　10H(音楽演奏中)
　　　　　　　　80H(ハードエラー)
　　　　　BX　=残りのセクタ数
　　　　　CX　=エラー情報(AHが80Hの場合)

説　明　CDの指定時間位置から，BXに指定されたセクタ数分だけデータを読み出し，DS:DIで示されるバッファに格納します．もどり値のBXには残りのセクタ数が格納されています．

データの読み取り時にエラーが起こった場合，BXに残りのセクタ数が返されます．

CD-ROM	INT 93H
データの読み取り(時間指定)〈拡張〉	機能コード15H

エントリ　AH　=15H
　　　　　AL　=デバイス番号(C0H)
　　　　　CH　=FFH
　　　　　CL　=分
　　　　　DH　=秒
　　　　　DL　=フレーム
　　　　　BX　=読み取りするセクタ数
　　　　　SI　=バッファアドレス(上位2バイト)
　　　　　DI　=バッファアドレス(下位2バイト)

リターン　AH　=00H(正常終了)
　　　　　　　　02H(デバイス番号エラー)
　　　　　　　　10H(音楽演奏中)
　　　　　　　　80H(ハードエラー)
　　　　　BX　=残りのブロック数
　　　　　CX　=エラー情報(AHが80Hの場合)

説　明

「データの読み取り(時間指定)(機能コード15H，CH＝00H)」では，アドレス空間は最大1MBまでに限定されますが，この拡張オペレーションでは，32ビットのアドレス空間に対応し，4GBまでのメモリをバッファとして使用できます．

CHに指定する値がFFとなり，SIとDIでバッファアドレスを物理アドレスで指定する点以外は，「データの読み取り(時間指定)(機能コード15H，CH＝00H)」と同様です．

CD-ROM　INT 93H
音楽演奏スタート　機能コード50H

エントリ

AH	＝50H
AL	＝デバイス番号(C0H)
CH	＝00H
CL	＝01H(時間指定)
DS:DI	＝音楽演奏データのアドレス

リターン

AH	＝00H(正常終了)
	02H(デバイス番号エラー)
	10H(音楽演奏中)
	80H(ハードエラー)
CX	＝エラー情報(AHが80Hの場合)

説　明

音楽演奏を開始します．

音楽演奏データの形式を示します．

(DS:DI)

0	E	B	演奏開始時間(分)
1	E	B	演奏開始時間(秒)
2	E	B	演奏開始時間(フレーム)
3	E	B	演奏終了時間(分)
4	E	B	演奏終了時間(秒)
5	E	B	演奏終了時間(フレーム)

演奏開始時間と終了時間を分，秒，フレームで指定します．

最後の曲を演奏する場合には，終了時間にトータル演奏時間より１フレーム少ない値を設定します．このオペレーションを実行すると，演奏開始とともにただちにリターンします．また，次のCD-ROM BIOSオペレーションを使用するためには，演奏を終了していなければなりません．演奏を途中で終了させるには，「音楽演奏ストップ(機能コード52H)」を使用します．

CD-ROM	INT 93H
音楽演奏スタート(リピート機能)〈拡張〉	機能コード50H

エントリ

AH	=50H
AL	=デバイス番号(C0H)
CH	=FFH
CL	=01H(時間指定)
DS:DI	=音楽演奏データのアドレス

リターン

AH	=00H(正常終了)
	02H(デバイス番号エラー)
	10H(音楽演奏中)
	80H(ハードエラー)
CX	=エラー情報(AHが80Hの場合)

説　明

「音楽演奏スタート(機能コード50H，CH=00H)」にリピート機能を加えたもので，演奏開始時間から演奏終了時間まで繰り返して演奏します．

放っておくと永久に止まらないので，止めたいときは「音楽演奏ストップ(機能コード52H)」を実行します．

CHに入力する値がFFHとなる以外は，「音楽演奏スタート(機能コード50H，CH=00H)」と同じです．

CD-ROM　　INT 93H

音楽演奏スタート(回数指定のあるリピート機能)〈拡張〉　機能コード50H

エントリ	AH	=50H
	AL	=デバイス番号(C0H)
	BH	=リピート回数
	CH	=FEH
	CL	=01H(時間指定)
	DS:DI	=音楽演奏データのアドレス
リターン	AH	=00H(正常終了)
		02H(デバイス番号エラー)
		10H(音楽演奏中)
		80H(ハードエラー)
	CX	=エラー情報(AHが80Hの場合)

説　明

「音楽演奏スタート(機能コード50H，CH=00H)」に回数指定のできるリピート機能を加えたものです．BHには，演奏を繰り返す回数より1少ない値を指定します．リピート回数をBHに指定することと，CHに入力する値がFEHとなる以外は，「音楽演奏スタート(機能コード50H，CH=00H)」と同じです．

CD-ROM　　INT 93H

音楽演奏情報の読み取り　機能コード51H

エントリ	AH	=51H
	AL	=デバイス番号(C0H)
	CH	=00H
	CL	=01H(時間指定)
	DS:DI	=音楽演奏データのアドレス
リターン	AH	=00H(正常終了)
		02H(デバイス番号エラー)
		80H(ハードエラー)
	CX	=エラー情報(AHが80Hの場合)

説　明　「音楽演奏スタート(機能コード50H)」で設定した内容を読み出し，DS:DIで指定した領域に格納します．

音楽演奏データの形式を示します．

(DS:DI)

0	E	B	演奏開始時間(分)
1	E	B	演奏開始時間(秒)
2	E	B	演奏開始時間(フレーム)
3	E	B	演奏終了時間(分)
4	E	B	演奏終了時間(秒)
5	E	B	演奏終了時間(フレーム)

CD-ROM　INT 93H
音楽演奏ストップ　機能コード52H

エントリ
AH　=52H
AL　=デバイス番号(C0H)
CH　=00H

リターン
AH　=00H(正常終了)
　　02H(デバイス番号エラー)
　　80H(ハードエラー)
CX　=エラー情報(AHが80Hの場合)

説　明　音楽演奏を終了し，CDドライブを停止させます．

音楽演奏を終了させた後，CDドライブを停止させずにデータを読み取るときは，「音楽演奏一時停止(機能コード55H)」を使用します．

CD-ROM	INT 93H
CDドライブ停止時間の設定〈拡張〉	機能コード52H

エントリ
AH　=52H
AL　=デバイス番号(C0H)
CH　=FFH
CL　=ドライブ停止時間

リターン
AH　=00H(正常終了)
　02H(デバイス番号エラー)
　80H(ハードエラー)
CX　=エラー情報(AHが80Hの場合)

説　明
CDドライブを動作させるオペレーションを実行し，それが終了したときに，自動的に停止するまでの時間(単位は秒)を設定します．

このファンクションを実行して，次のCD-ROM BIOSアクセス後に有効となります．

0を設定した場合は，ドライブは通常停止しません．初期値は約20秒です．

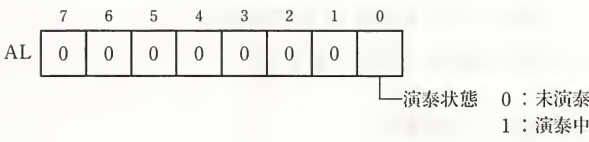
CD-ROM	INT 93H
音楽演奏状態の読み取り	機能コード53H

エントリ
AH　=53H
AL　=デバイス番号(C0H)
CH　=00H
CL　=00H(サブコードQデータ)
DS : DI　=音楽演奏状態データのアドレス

リターン
AH　=00H(正常終了)
　02H(デバイス番号エラー)
　80H(ハードエラー)
AL　=演奏状態(0：演奏してない，1：演奏中)
CX　=エラー情報(AHが80Hの場合)

説　明　CDの演奏状態を調べます.

演奏状態データを格納する領域の先頭アドレスをDS:DIで指定して，このオペレーションを実行すると，演奏状態を知ることができます．エントリのCLの00Hは，BIOSに対して演奏状態を返すように指示するものです.

演奏の有無はALに返されます．ALのデータの形式を示します.

音楽演奏状態データの形式を示します.

(DS:DI)

0	R	B	予約済
1	R	B	演奏曲番号
2	R	B	予約済
3	R	B	トラック内演奏時間(分)
4	R	B	トラック内演奏時間(秒)
5	R	B	トラック内演奏時間(フレーム)
6	R	B	予約済
7	R	B	ディスク内演奏時間(分)
8	R	B	ディスク内演奏時間(秒)
9	R	B	ディスク内演奏時間(フレーム)

ここで，トラックとは物理的に連続したデータのかたまりであり，オーディオ用のCDでいえば，1つのトラックが1つの曲に対応しています.

CD-ROM	INT 93H
CD情報の読み取り	機能コード54H

エントリ
AH　=54H
AL　=デバイス番号(C0H)
CH　=00H
CL　=00H(TOC-Table of contents)
DS:DI　=CD情報データのアドレス

リターン
AH　=00H(正常終了)
02H(デバイス番号エラー)
10H(音楽演奏中)
80H(ハードエラー)
CX　=エラー情報(AHが80Hの場合)

説　明

CDの詳細な情報を調べます.

CDの情報を格納する領域の先頭アドレスをDS:DIで指定して,このオペレーションを実行すると,CDのTOC(Table of Contents-CDに収容されている曲数,各トラック(曲)の演奏開始時間位置(分,秒,フレーム)など)を知ることができます.エントリのCLの00Hは,BIOSに対してTOCを返すように指示するものです.

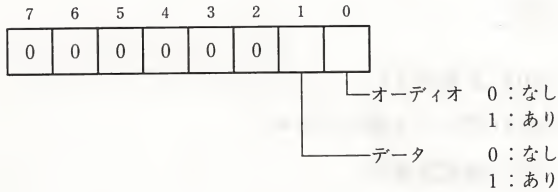
CD情報データの形式を示します.

(DS:DI)

0	R	B	コンパクトディスクタイプ
1	R	B	先頭トラック(曲)番号
2	R	B	最終トラック(曲)番号
3	R	B	ディスク内演奏時間(分)
4	R	B	ディスク内演奏時間(秒)
5	R	B	ディスク内演奏時間(フレーム)
6	R	B	トラック演奏開始時間(分)
7	R	B	トラック演奏開始時間(秒)
8	R	B	トラック演奏開始時間(フレーム)
9	R	B	トラック演奏開始時間(分)
10	R	B	トラック演奏開始時間(秒)
11	R	B	トラック演奏開始時間(フレーム)
			最終トラックまで続く

このTOCのデータ領域は，最大303バイト必要です．

ここで，トラックとは物理的に連続したデータのかたまりであり，オーディオ用のCDでいえば，１つのトラックが１つの曲に対応しています．

コンパクトディスクタイプの形式を示します．

トラック演奏開始時間(分)の７ビット目がONの場合は，そのトラックはデータトラックです．

CD-ROM	INT 93H
音楽演奏一時停止	機能コード55H

エントリ

AH　＝55H
AL　＝デバイス番号(C0H)
CH　＝00H

リターン

AH　＝00H(正常終了)
02H(デバイス番号エラー)
22H(すでに一時停止中である)
80H(ハードエラー)
CX　＝エラー情報(AHが80Hの場合)

説　明

音楽演奏を一時停止します．
再開は，「音楽演奏一時停止解除(機能コード56H)」の実行によります．

CD-ROM	INT 93H
音楽演奏一時停止解除	機能コード56H

エントリ	AH	＝56H
	AL	＝デバイス番号(C0H)
	CH	＝00H
リターン	AH	＝00H(正常終了) 02H(デバイス番号エラー) 10H(音楽演奏中) 23H(一時停止中でない) 80H(ハードエラー)
	CX	＝エラー情報

説　明

「音楽演奏一時停止(機能コード55H)」の実行後，演奏を再開します．
一時停止中以外のときに実行するとエラーになります．

第8章

キーボードBIOS

この章では，キーボードBIOSについて解説します．キーボードBIOSは，FMRシリーズのものと共通の仕様となっています．なお，FMRシリーズでは，キーボードBIOSでマウスの制御が可能ですが，Towns OS上では，TOWNSマウスはこれに対応しておらず，マウスBIOSで制御します．

8.1 キーボードBIOSの概要

FMTOWNSでは，キー配置の異なる4種類のキーボード(親指シフト，JIS，それぞれテンキー付とテンキーなし)が用意されており，いずれかを選択して使用することができます．

キーボードから入力されたデータは，各種コードに変換された後，いったんキーボードバッファに格納され，プログラムからの読み出し要求に従って通知されます．このとき，インタフェース側からは，入力された文字を自動的にエコーバック(折り返し表示)することはありません．

変換コード系には，スキャンコード，ASCIIコード，JISコードの3種類があります．ASCIIコードとJISコードでは，詳細情報としてキーアドレスと，CTRL，SHIFTキーなどのシフトキー情報が合わせて通知されるので，より細かい制御ができます．

また，キーボードからの特定の文字コードごとに，別の文字や文字列を割り当てる機能もあり，オペレータの入力負荷を軽減することができます．

8.2　キーボードBIOS一覧

表II-8-2にキーボードBIOSの一覧を示します．

▼表II-8-2　キーボードBIOS一覧

機能名称	機能コード
初期化	00H
バッファリング機能の設定	01H
コード系の設定	02H
コード系の読み取り	03H
キーボードロックの制御	04H
クリック音の制御	05H
バッファのクリア	06H
入力のチェック	07H
シフトキー状態の読み取り	08H
文字の読み出し	09H
マトリクス入力	0AH
入力文字列の追加	0BH
PFキー割り込み処理ルーチンの登録	0CH
PFキー割り込み処理ルーチンの読み取り	0DH
キー割り当て	0EH
キー割り当て状態の読み取り	0FH

8.3　キーボードBIOSの基本機能と用語

ここでは，キーボードBIOSの基本的な機能や用語について解説します．

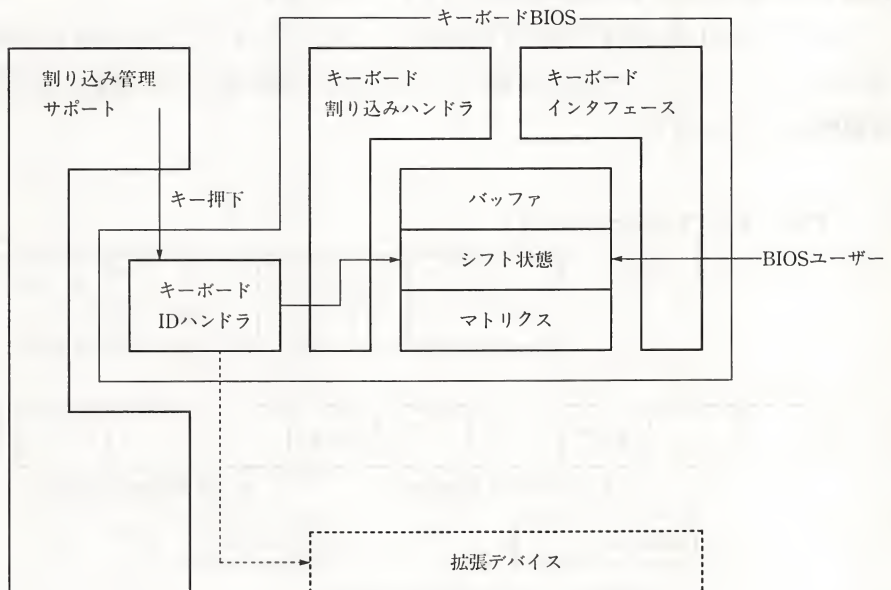
●キーボードBIOSとキー入力

キーボードBIOSは，キー押下などによって起動され，ユーザーからの要求と同期して，入力情報を通知します．このために割り込みが使われていますが，ユーザーは単にBIOSをコールするだけですむようになっています．

一連の動作を段階別に説明すると次のようになります．

①ユーザーからのBIOSコールがあったとき，引き渡しすべきデータがあるかどうかを調べ，あるときはただちに引き渡しする．ないときは，その情報が登録され，実行中のプログラムは待ち状態になる．
②キーの押下とともに割り込みが発生し，割り込み管理BIOSが働く．このとき，内部のキーボードIDハンドラが起動される．
③キーボードIDハンドラはキーボードからの割り込みであるかどうかを検査し，該当する場合は，さらにキーボード割り込みハンドラを起動する．
④キーボード割り込みハンドラは，押下されたキーのアドレスを読み取り、必要があれば指定されたコード体系に従って変換する．その結果は，キーボードバッファに格納される．このときCTRL，SHIFTなどのシフトキーを含めたすべてのキーの押下情報を記憶する．
⑤ユーザーに引き渡しする条件が整った段階で，④のデータが引き渡しされる．

キーボードBIOSの概念図を示すと，図II-8-1のようになります．

▼図II-8-1　キーボードBIOSの概念図

●キーの種別

キーボード上の各キーは次のようにグループ分けされています．それぞれのグループ単位で入力を禁止(マスク)することが可能です．

文字キー　　：以下のキー種別に示されたキー以外の文字キー

PF(programmable Function)キー

：PF(1～20)，取消，実行，漢字辞書，単語抹消，単語登録，前行，次行，半角／全角，かな漢字，変換，無変換

編集キー　　：↑，↓，←，→，削除／EL，挿入／DUP，HOME/CLS

通信キー　　：他システムとの通信用のキーであり，現在は未サポートです．

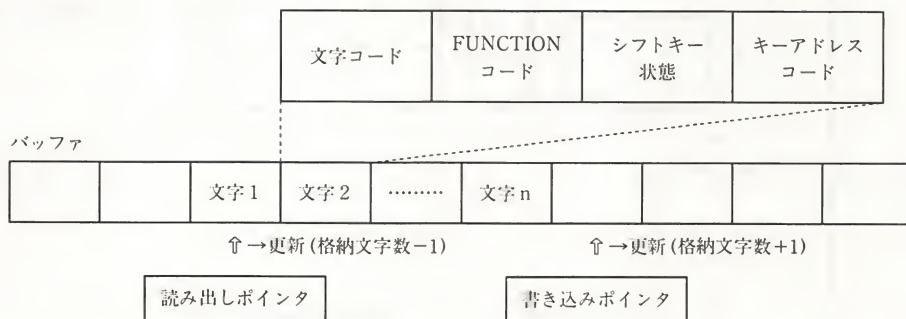
SF(System Function)キー

：BREAK，COPY

●キーボートバッファ

キーボードからの入力データは，128文字分のキーボードバッファを経由して渡されます．バッファは，入力した文字データを蓄積する働きを持つ一方，ユーザーに引き渡し済みの部分の内容を実質的に消去する(参照範囲外とする)ことができます．

このために図II-8-2のような，書き込みポインタ(入力された文字を格納する場所を示す)，読み出しポインタ(ユーザーに引き渡しする文字の存在場所を示す)，格納文字数を収納する作業領域を持っています．

▼図II-8-2　キーボードバッファ

(1)バッファリングを行わない場合

最も考えやすいのは，バッファのデータ蓄積機能を使わず，1文字ずつ入力するパターンです．

この動作モードでは，書き込みポインタと読み出しポインタは常に同じ位置を指しており，入力された文字を格納する場所とユーザーが読み出す場所とが，メモリ上で一致しています．

このため，以前の文字が読み出されないうちに次の文字が入力されたときは，以前の文字データは消えてしまいます．

(2)バッファリングを行う場合

各ポインタの本来の機能を生かした動作モードです．

このモードでは，１文字書き込みを行うたびに書き込みポインタの値が１文字分だけ増え，１文字読み出しが行われる都度，同様に読み出しポインタが更新されます．また，書き込みの際は格納文字数が１加算され，読み出しのときは１減算されるようになっているので，常に格納文字数の値は，読み出されていない文字の数を示しています．そして格納文字数が０のときは読み出しが行われないので，読み出しポインタが書き込みポインタに先行することはありません．バッファはリング構造になっており，最後のポイントは先頭のポイントに連結しています．

もし，読み出しが行われないまま書き込みが続いて，リングをひと回りしてしまったときは，そのまま続行すると以前のデータが消え，格納文字数の値と矛盾が生じます．従って，読み出しによってバッファが空くまで，BIOSは新規に入力された内容を無視します．

１．文字コード

キーボードBIOSでは，スキャンモード(キーのアドレスを読み取る)とエンコードモード(文字コードに変換して読み取る)をサポートしています．また，エンコードモードでは，ASCIIまたは，JISのいずれかの文字コード体系が選択できます．

(1)スキャンモード

ハードウェアに近い読み出しモードです．このモードでは，押下されたキーのアドレスが読み出され，またそのキーが押されている(MAKEという)か，離された(BREAKという)かの情報も与えられます．

このモードを利用すると，同じ文字でもキーが異なる(例えば数字はテンキーで入力されることもある)場合，どちらから入力されたかを識別することも可能です．

(2)エンコードモード

キーのアドレスに対応した文字コードに変換して読み取るモードで，一般に使われているのがこの形態です．

ASCIIコードは７ビットで形成され，最上位の１ビットは常に０になっています．英文字専用のコード体系のため，カナ文字を扱うことができず，カナモードへの移行操作は無視されます．また，カナ文字以外でも８ビットになる文字キー(〝「〟など)は無効です．

JISコードは，８ビットのコード体系で，英文字の外にカナ文字を加えたものです．通常はこのモードが使われます．漢字(２バイト文字)は，シフトJIS漢字コードで扱われています．

エンコードモードでは，PFキーを入力したとき，BIOS独自の内部コードが与えられます．

2．FUNCTION コード

読み出しモードや変換コード体系などの情報です．BIOS のオペレーションを通じて参照することができます．詳しくはコード系の読み取りオペレーションなどを参照してください．

3．シフトステータス

現在有効なシフトキーなどの状態を参照するための情報です．項目は表II-8-1のとおりで，シフトキー状態の読み取りオペレーションで参照することができます．

▼表II-8-1　シフトキー状態とビットの値

シフトキー状態	ビットの値（0：OFF，1：ON）
英大／英小(英大のとき1) カナ(カナのとき1)	英大のとき1，カナのとき1となる．ただし，カナが1のときはカナモードが優先される．
SHIFT CTRL	各キーを押しながら，文字入力したときに1となる．
右親指シフト 左親指シフト	親指シフトキーを同時打鍵して文字入力したときに1となる．

右，あるいは左親指シフトの情報がいずれか1であるときには，親指シフトキーの同時打鍵により文字が入力されたことを示しており，親指シフトキーボードが接続されているときのみ有効です．JIS キーボードでは常に0が通知されます．

4．キーアドレスコード

ハードウェアがスキャンして得たキーアドレスの値そのままが，記憶されます．文字の読み出しオペレーションで参照することができます．

●文字列の割り当て

文字コードの指定とは別に，文字コード単位に文字列を割り当てることができます．すなわち，ある1文字が入力されたとき，登録済みの文字列の各文字が連続して入力されたかのように制御することが可能です．

このことを利用して，例えば，PF キーが押されたとき“dir”の3文字が入力されたとみなすなどのようにすると，コマンドや決まりきった文字列の入力を合理化できます．

●入力モード

入力モードは，親指シフトキーボードと JIS キーボードの場合で，異なるシフト変遷をします．

8.4　キーボードBIOSリファレンス

キーボードBIOSについて個別に詳しく解説します．

キーボード　INT 90H
初期化　機能コード00H

エントリ　AH　＝00H

リターン　AH　＝00H（正常終了）

説　明　キーボードを次のように初期化します．

内　容	状　態
バッファリング機能	あり
キーボードバッファ	クリア状態
コード系	JIS(8ビット)コード
キーのマスク	なし
シフト状態	英小文字
キーボードロック	なし
クリック音	あり
PFキー割り込み機能	なし
キー割り当て	キー配列どおり
オートリピート待ち時間	400ms
オートリピート周期	30ms

コード系が"JIS(8ビット)コード"の状態では，漢字はシフトJIS漢字コードで扱われます．

キーボード　INT 90H
バッファリング機能の設定　機能コード01H

エントリ　AH　＝01H

AL　＝00Hまたは02H以上（バッファリング機能あり）

01H（バッファリング機能なし）

リターン　AH　＝00H（正常終了）

説　明　キーボードから文字入力の際，バッファリングを行うか否かを設定します．

ALレジスタに02H以上の値を設定した場合は，バッファリング機能ありと見なされます．なお，バッファリング機能の設定後，キーボードバッファはクリア状態(格納文字数＝0)となりますが，その外の状態は変更されません．

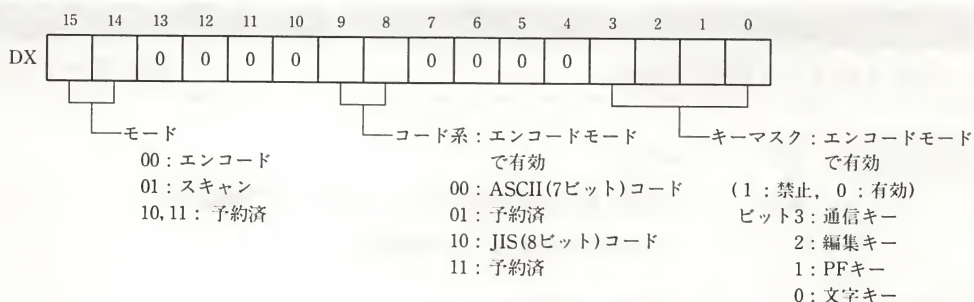
内　容	状　態
バッファリング	指定値
キーボードバッファ	クリア状態
コード系	変更なし
キーのマスク	変更なし
シフト状態	変更なし
キーボードロック	変更なし
クリック音	変更なし
PFキー割り込み機能	変更なし
キー割り当て	変更なし
オートリピート開始時間	変更なし
オートリピート周期	変更なし

キーボード　INT 90H
コード系の設定　機能コード02H

エントリ　AH　＝02H

DX　＝モード／コード系／キーマスク

リターン　AH　＝00H(正常終了)

説　明　「入力のチェック(機能コード07H)」，「文字の読み出し(機能コード09H)」で文字コードを通知する際にモード／コード系を指定します．指定形式を示します．

コード系が"JIS(8ビット)コード"の状態では，漢字はシフトJIS漢字コードで扱われます．

なお，未サポートキー(BIOSでサポートしているが，ハードウェア的にないキー)のマスクは，無条件に禁止に設定されます．

コード系を切り換えた後は，キーボードバッファはクリア状態に，シフト状態は英小文字に設定されますが，その外の状態は変更されません．

内容	状態
バッファリング機能	変更なし
キーボードバッファ	クリア状態
コード系	指定値
キーのマスク	指定値
シフト状態	英小文字
キーボードロック	変更なし
クリック音	変更なし
PFキー割り込み機能	変更なし
キー割り当て	変更なし
オートリピート開始時間	変更なし
オートリピート周期	変更なし

キーボード　INT 90H

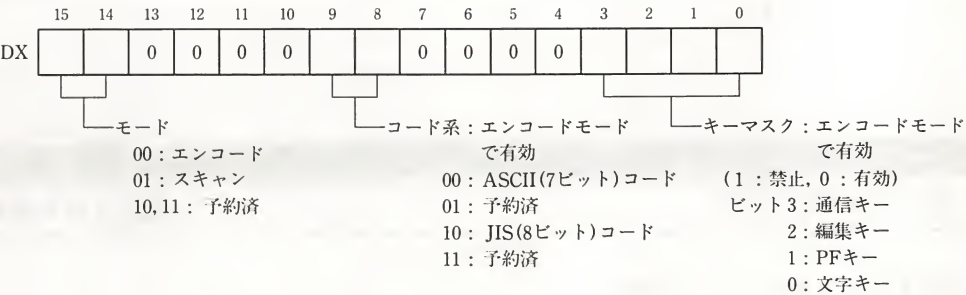
コード系の読み取り　機能コード03H

エントリ　AH　＝03H

リターン　AH　＝00H(正常終了)

DX　＝モード／コード系／キーマスク

説　明　現在のコード系を通知します．

通知形式を次に示します．

コード系が"JIS(8ビット)コード"の状態では，漢字はシフトJIS漢字コードで扱われます．

なお，未サポートキー(BIOSでサポートしているが，ハードウェア上にないキー)のマスクは無条件に禁止に設定されます．

キーボード　INT 90H

キーボードロックの制御　機能コード04H

エントリ　AH　＝04H

AL　＝00Hまたは02H以上(キーボードロックなし)
01H(キーボードロックあり)

リターン　AH　＝00H(正常終了)

説　明　エンコードモードのとき，キーボードよりキーの入力を許すかどうかを制御します．

ALレジスタに02H以上の値を指定した場合には，キーボードロックなしと見なされます．

キーボード　INT 90H

クリック音の制御　機能コード05H

エントリ　AH　＝05H

AL　＝00Hまたは02H以上(クリック音あり)
01H(クリック音なし)

リターン　AH　＝00H(正常終了)

説　明　キー押下時(シフトキーを除く)にクリック音を鳴らすかどうかを制御します．

ALレジスタに02H以上の値を指定した場合には，クリック音ありと見なされます．

キーボード　INT 90H

バッファのクリア　機能コード06H

エントリ　AH　＝06H

AL　＝00H

リターン　AH　＝00H(正常終了)

説　明　キーボードバッファ内のすべての文字を消去します．

キーボード	INT 90H
入力のチェック	**機能コード07H**

エントリ　AH　=07H

リターン　AH　=00H(正常終了)
AL　=文字数
DH　=FFH(入力文字なし)
　　FFH以外の値(入力文字あり)
DL　=文字コード(DHがFFH以外の場合に有効)
BX　=エンコード情報(BH：キーアドレス，BL：シフトキー状態)

説　明

キーボードバッファ内に格納されている文字数と，キーボードバッファの読み出しポインタが示している文字(文字コードおよびエンコード情報)を通知します．

その際，読み出しポインタは更新されません．

●文字コード

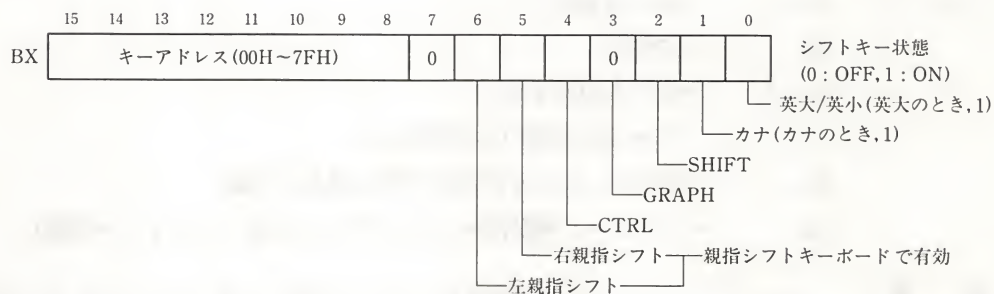
文字コードは，モードにより通知形式が次のように異なります．

エンコードモードの場合
DH　=00H(PFキー，シフトキー以外のキー押下により文字コードが発生したことを示す)
　　80H(PFキー押下により文字コードが発生したことを示す)
　　FFH(入力文字がないことを示す)
DL　=文字コード(DHがFFH以外のときに有効)

スキャンモードの場合
DH　=00H(スキャンコードが有効)
　　FFH(入力文字がないことを示す)
DL　=スキャンコード

キーアドレスは各機種固有の形式です．

●エンコード情報

エンコード情報はエンコードモードにおいて有効であり，次のようにどのキーから文字コードが得られたかを示すキーアドレスと，押下時のシフトキー状態から構成されます．

なお，キーボードバッファ内に格納されている文字が，次のようにして入力されたとき，キーアドレス(BH)には，00Hが通知されます．

- 「入力文字列の追加(機能コード 0BH)」により入力された文字
- 「キー割り当て(機能コード 0EH)」により入力された文字
- DSR シーケンス(ESC [6n)による，CPR シーケンス(ESC [Pl；Pc R)(カーソル位置)の通知
- ESC　?シーケンスによるカーソル位置の通知

文字数はキーボードバッファに格納されている文字(この場合，キーに2文字以上の文字が割り当てられていたとしても，1文字と見なされます)，およびキー割り当て処理時に使用するキー割り当てバッファ(1文字)です．

キーボード　INT 90H

シフトキー状態の読み取り　機能コード08H

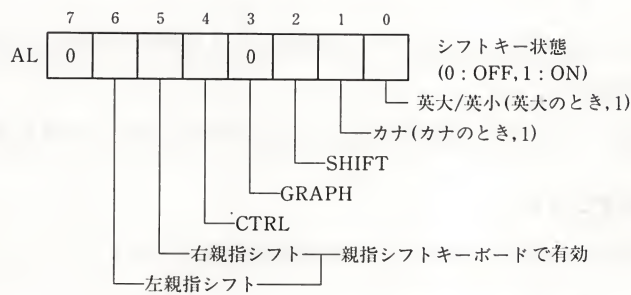
エントリ　AH　＝08H

リターン　AH　＝00H(正常終了)

AL　＝シフトキー状態

説　明　現在のキーボードのシフトキー状態を通知します．

スキャンモードのときには意味を持ちません．

通知形式を示します．

キーボード　INT 90H

文字の読み出し　機能コード09H

エントリ　AH　＝09H

AL　＝00H(入力待ち)

　01H(入力待ちなし)

リターン　AH　＝00H(正常終了)

DH　＝FFH(入力文字なし)

　FFH以外の値(入力文字あり)

DL　＝文字コード(DHがFFH以外の場合に有効)

BX　＝エンコード情報(BH：キーアドレス，BL：シフトキー状態)

説　明　キーボードから文字を入力します．

すなわち，キーボードバッファから読み出しポインタが示す文字(文字コードとエンコード情報)を取り出して通知します．「入力のチェック(機能コード07H)」と異なり，文字を取り出した後，読み出しポインタが更新されます．

入力待ちが指定された場合,有効なキー(コード系あるいはキーのマスクにより無効なキーがあります)が押下されるまで待ちます.

入力待ちなしが指定されていて，かつ，この機能が呼び出された際，まだ文字が入力されていなければ，直ちにこの機能から復帰します．その際には次に示す情報が通知されます.

AH＝00H
DH＝FFH
DL＝FFH
BH＝FFH
BL＝この機能が呼び出された際のシフトキー状態

すでに文字が入力されている場合は，入力待ちが指定された場合の機能と同じです.

ALレジスタに02H以上の値を設定した場合には入力待ちと見なされます.

●文字コード

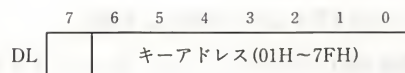
文字コードは，モードにより通知形式が異なります.

エンコードモードの場合

DH　＝00H(PFキー，シフトキー以外のキー押下により文字コードが発生したことを示す)
　　80H(PFキー押下により文字コードが発生したことを示す)
　　FFH(入力文字がないことを示す)
DL　＝文字コード(DHがFFH以外の場合に有効)

スキャンモードの場合

DH　＝00H(スキャンコードが有効)
　　＝FFH(入力文字がないことを示す)
DL　＝スキャンコード

	7	6 5 4 3 2 1 0
DL		キーアドレス(01H～7FH)

└0：キーアドレスで通知されたキーが押されていることを示す(MAKE).
　1：キーアドレスで通知されたキーが離されたことを示す(BREAK).

キーアドレスは各機種固有の形式です.

●エンコード情報

エンコード情報はエンコードモードにおいて有効であり，どのキーから文字コードが得られたかを示すキーアドレスと押下時のシフトキー状態から構成されます．

なお，キーボードバッファ内に格納されている文字が次のようにして入力されたとき，キーアドレス(BH)には00Hが通知されます．

- 「入力文字列の追加(機能コード 0BH)」により入力された文字
- 「キー割り当て(機能コード 0EH)」によりに入力された文字
- DSR シーケンス(ESC [6n)による，CPR シーケンス(ESC [Pl ; Pc R)(カーソル位置)の通知
- ESC　?シーケンスによるカーソル位置の通知

キーボード	INT 90H
マトリクス入力	機能コード0AH

エントリ　AH　＝0AH
DS：DI　＝マトリクス格納領域アドレス

リターン　AH　＝00H（正常終了）

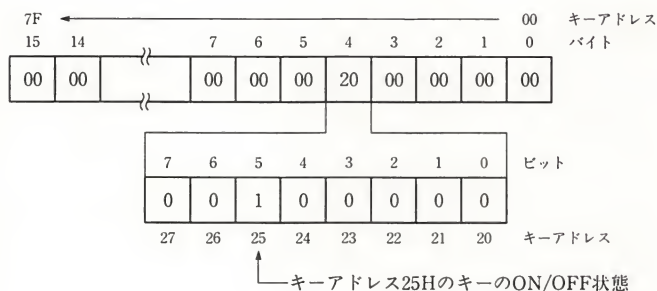
説　明

キーボードのすべてのキーの押下状態を128ビット（16バイト）の領域に通知します．

マトリクス格納領域の形式は次のとおりです．

キーとその状態の格納位置は，次に示す計算式によって求められ，その対応ビットが1のときは，キーがMAKE（キーが押されている）状態にあり，0のときはBREAK（キーが離されている）状態にあることになります．

（格納領域先頭からの相対アドレス）＝（キーアドレス）÷ 8
（ビット位置）＝（キーアドレス） mod 8（8で割った余り）

例えば，キーアドレス25Hのキーが押されている状態では，次のようになります．

キーアドレスに対応するキーがない場合には，0が通知されます．

キーボード　INT 90H

入力文字列の追加　機能コード0BH

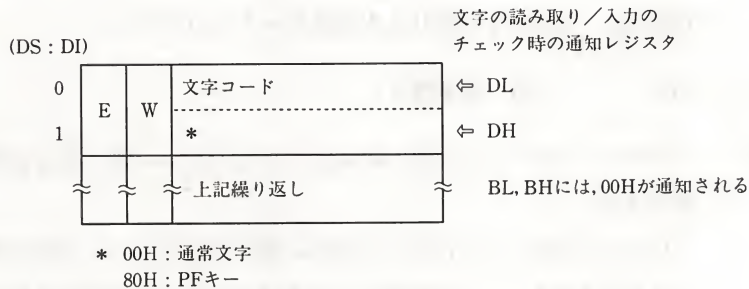
エントリ	AH	＝0BH
	AL	＝00Hまたは，02H以上（キーボードバッファの先頭に挿入－読み出しポインタの前側） 01H（キーボードバッファの最後に追加－書き込みポインタ以降）
	CX	＝追加文字数
	DS : DI	＝文字列アドレス
リターン	AH	＝00H（正常終了） 02H（入力文字列の追加は不可）

説　明

キーボードバッファの先頭あるいは最後に文字列を追加します．

このオペレーションのあとで，「文字の読み出し（機能コード09H）」を使用してみると，先頭に挿入した場合は，追加した文字列を最初に読み取り，続いてすでにキーボードから入力していた文字を読み取ります．

最後に追加した場合には，すでにキーボードから入力した文字をすべて読み取った後，その追加文字を読み取ります．

文字列の設定形式を示します．

この機能は，「バッファリング機能あり」の場合に有効であり，「バッファリング機能なし」の場合には，AHに02H（入力文字列の追加は不可）が通知されます．

ALレジスタに02H以上の値を設定した場合は，AL＝0と見なされます．

追加した文字は，「入力のチェック(機能コード07H)」，「文字の読み出し(機能コード09H)」を使用した場合には，キー割り当て(機能コード0EH)により割り当てられた文字に変換され通知されます．すなわち，オペレータから入力された文字となんら変わりなく扱われます．

文字格納形式の詳細は「入力チェック(機能コード07H)」あるいは，「文字の読み出し(機能コード09H)」を参照してください．

ただし，キーボードバッファには，指定された文字コードを単に格納するのみで，格納した文字コードが正常であるか否かはまったくチェックしないので注意してください．

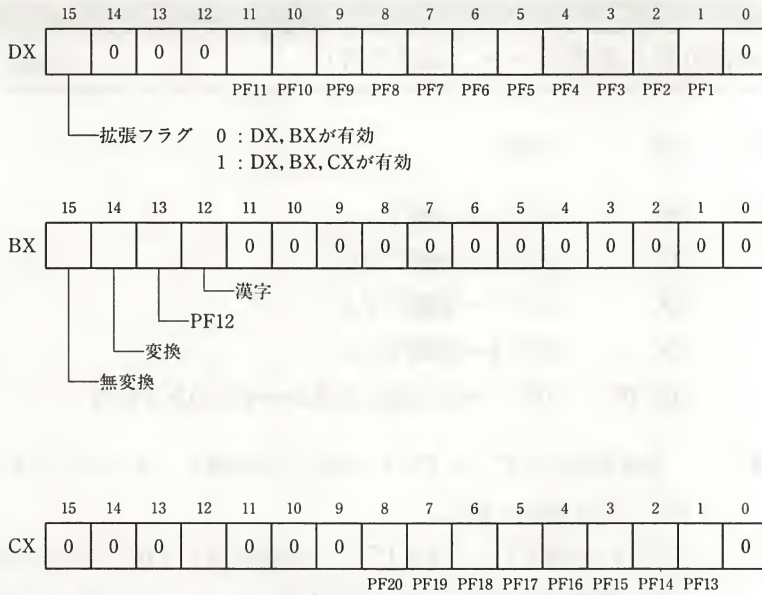
キーボードバッファの空き領域がなく，追加が不可の場合には，エラーを通知するのみで，キーボードバッファの内容は変化しません．

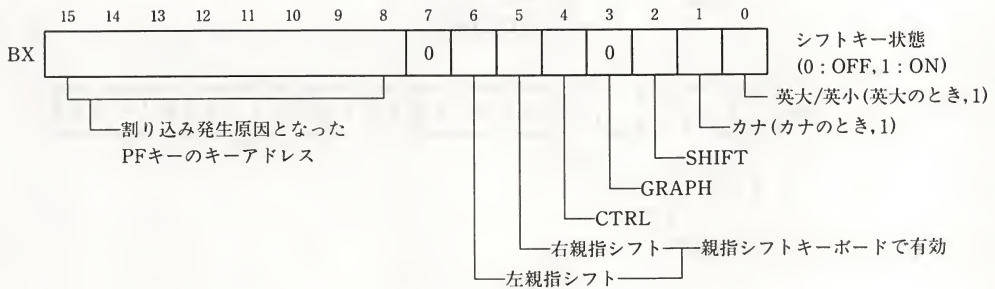
キーボード	INT 90H
PFキー割り込み処理ルーチンの登録	**機能コード0CH**

エントリ	AH	＝0CH
	DX	＝PFキー制御フラグ
	BX	＝PFキー制御フラグ
	CX	＝PFキー制御フラグ
	DS : DI	＝PFキー割り込み処理ルーチンのアドレス
リターン	AH	＝00H(正常終了)

説　明

PFキーが押下された際に呼び出される，PFキー割り込み処理ルーチンを登録します．

PFキー制御フラグはビット単位に個別に各PFキーに割り当てられ，各フラグの示す値が1のときに割り込み処理ルーチンが呼び出されます．0の場合には，通常の文字キーと同様にPFキーに対応する文字コードとエンコード情報がキーボードバッファに格納されます．さらに，拡張フラグを1にするとCXが有効になります．

PF キー割り込み処理ルーチンを呼び出す際には，セグメント間呼び出し(far CALL)が使用されるので，セグメント間復帰(far RET)を使用してキーボード割り込みハンドラに復帰してください(レジスタを復旧する必要はありません).

呼び出し時には，BX レジスタにより割り込み発生原因となったキーアドレスと，シフトキー情報がハンドラに通知されます.

PF キーがマスクされている場合でも，設定情報は保持されおり，マスクが解除された際に有効となります.

キーボード	INT 90H
PFキー割り込み処理ルーチンの読み取り	機能コード0DH

エントリ　AH　=0DH

リターン　AH　=00H(正常終了)
DX　=PFキー制御フラグ
BX　=PFキー制御フラグ
CX　=PFキー制御フラグ
DS:DI　=PFキー割り込み処理ルーチンのアドレス

説　明　現在登録されているPFキー割り込み処理ルーチンのアドレスとPFキー制御フラグを通知します.

PFキー制御フラグは各PFキーに割り当てられ，各ビットの示す値が1のときに割り込み処理ルーチンが呼び出されます. 0の場合には通常の文字キーと同様に,キーボードバッファにPFキーに対応する文字コードと,エンコード情報が格納されます. 拡張フラグが1のときにはCXが有効であることを表しています.

キーボード	INT 90H
キー割り当て	**機能コード0EH**

エントリ　AH　=0EH

AL　=00Hまたは02H以上(割り当てられた文字に押下されたキーアドレスを付加)

01H(割り当てられた文字のキーアドレスに00Hを付加)

DX　=文字コード

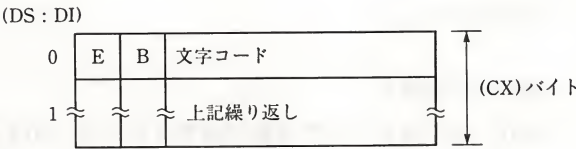
CX　=割り当て文字数

DS:DI　=文字列アドレス

リターン　AH　=00H(正常終了)

説　明　文字コードが示すキーに文字列を割り当てます.

キー押下時には，割り当てられる前の文字コードが，キーボードバッファに格納されます.

文字列の指定形式を示します.

AL=00Hであれば,エンコード情報の押下キーのキーアドレスと押下時のシフトキー状態がすべての文字列に付加されます.

AL=01Hであれば，エンコード情報のキーアドレスには00Hが，シフトキー状態には押下時のシフトキー状態が，すべての文字列に付加されます.

ALレジスタに20H以上の値を設定した場合には，AL=00Hと見なされます.

文字コードの指定形式を次に示します.

英数カナ，グラフィック文字キーの場合

DH　=00H

DL　=ASCII／JISコード

PFキーの場合

DH　=80H

DL　=PFキーコード

なお，割り当て可能な最大文字数はキー種別によって異なります．この文字数を越えたキー割り当てに対しては，先頭からの最大文字数分の文字列が有効となります．

PFキー	15文字
編集キー	7文字
コントロールキー以外の文字コード	7文字
コントロールキー(CTRL＋@～_)	割り当て不可

割り当て文字数に0を指定したときには，その文字コードに対応するキーの押下が読み取れなくなるので注意してください．

キーボード	INT 90H
キー割り当て状態の読み取り	**機能コード0FH**

エントリ
AH　＝0FH
DX　＝文字コード
DS：DI　＝文字列アドレス

リターン
AH　＝00H(正常終了)
AL　＝00H(割り当てられた文字に押下されたキーのキーアドレスを付加
　01H(割り当てられた文字のキーアドレスに00Hを付加
CX　＝割り当て文字数

説　明
文字コードが示すキーに割り当てられている文字列を通知します．
文字列の通知形式を示します．

CXレジスタで通知される文字数は，「キー割り当て(機能コード0EH)」によって割り当てられた文字数です．

領域サイズが，割り当てられた文字数よりも小さい場合には，割り当て文字列の先頭から領域サイズ分の文字が通知されます．

第 9 章

ディスクBIOS

この章では，ディスク BIOS について解説します．

ディスク BIOS により，フロッピィディスクドライブとハードディスクドライブを制御することができます．第 7 章で解説されている CD-ROM BIOS も，ディスク BIOS の 1 つですが，制御の仕方が著しく異なるため別章で解説しています．

9.1 ディスク BIOS 一覧

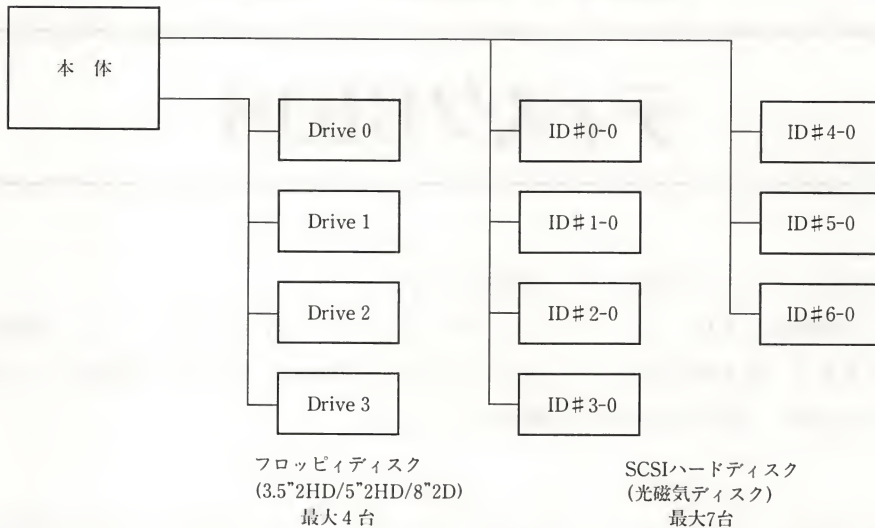
表II-9-1にディスク BIOS の一覧を示します．

▼表II-9-1 ディスク BIOS 一覧

機能名称	機能コード	備　考
ドライブモードの設定	00H	FD
ドライブモードの取り出し	01H	FD
ドライブステータス情報の取り出し	02H	FD，HD
シリンダ 0 へのシーク	03H	FD，HD
シーク	04H	FD
データの読み出し	05H	FD，HD
データの書き込み	06H	FD，HD
セクタの検査	07H	FD，HD
ハードディスクコントローラのリセット	08H	HD
セクタ ID の取り出し	09H	FD
トラックのフォーマット	0AH	FD
詳細エラー情報の取り出し	0DH	HD

FD　：フロッピィディスク
HD　：ハードディスク

図II-9-1にディスクBIOSがサポートする補助記憶周辺装置とその最大構成を示します.

▼図II-9-1　ディスクBIOSがサポートする補助記憶装置

注）FMTOWNSの内蔵ドライブは3.5インチ，拡張ドライブは5インチ.
3.5インチ2D，5インチ2Dはリードのみ可能.
SCSIハードディスクには20MB，40MB，60MB，130MBなどがある.
SCSIハードディスク(光磁気ディスク)は，以降，〝ハードディスク〟と略記する.

9.2　ディスクBIOSオペレーションの共通事項

ここでは，ディスクBIOSのオペレーションに共通する事項について解説します.

●デバイスとドライブ番号の識別

補助記憶装置をアクセスするオペレーションでは，ALにデバイスの種類とドライブ番号を指定します．図II-9-2にその形式を示します.

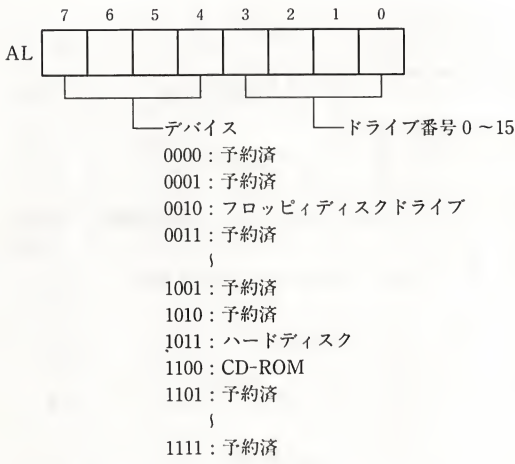
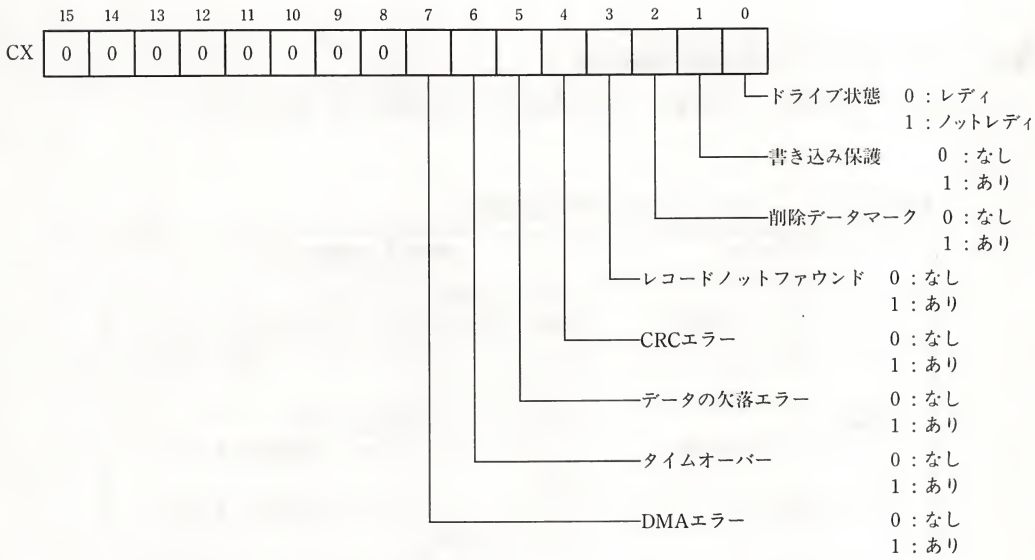
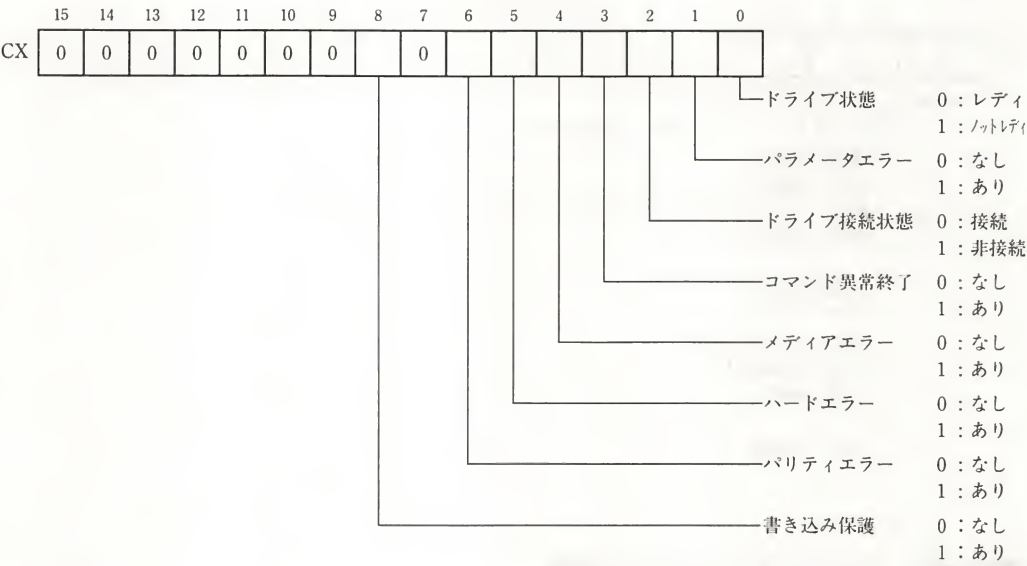
●エラー情報の識別

ハードエラー情報はCXに設定されます.

図II-9-3にフロッピィディスクのエラー情報の形式を，図II-9-4にハードディスクのエラー情報の形式を示します.

エラー発生時，BIOSではリトライを行いません.

なお，「ドライブステータス情報の取り出し(機能コード02H)」は，リトライの対象外です.

▼図II-9-2　デバイス番号の形式

▼図II-9-3　フロッピィディスクのエラー情報

▼図II-9-4 ハードディスクのエラー情報

●ハードディスクエラー発生時の処理方法

ハードディスクでエラーが発生した場合の処理の方法を表II-9-2に示します．

▼表II-9-2 ハードディスクのエラー処理方法

エラー内容	原因または対処方法
ノットレディ（ドライブ状態）	ドライブの電源が入っていない． 電源投入後30秒たってもレディにならない．
パラメーターエラー	指定されたコマンド，パラメータに誤りがある． プログラムミス．
非接続（ドライブ接続状態）	指定したドライブが存在しない． ハードディスクコントローラの電源が入っていない．
コマンド異常終了	ハードディスクがコマンドを異常終了させた． リトライが必要．
メディアエラー	メディアがキズなどによりアクセスができない． リトライが必要．
ハードエラー	コマンド実行中，ハード的にエラーが発生し，アクセスが中断された．ハードディスクリセットが必要．
パリティエラー	SCSIバス上でパリティエラーが発生した． リトライが必要．
書き込み保護	光磁気ディスク媒体が書き込み保護されている．

リトライは，シリンダ0へシーク(INT 93H AH=03H)後，行ってください．

9.3　ディスクBIOSリファレンス

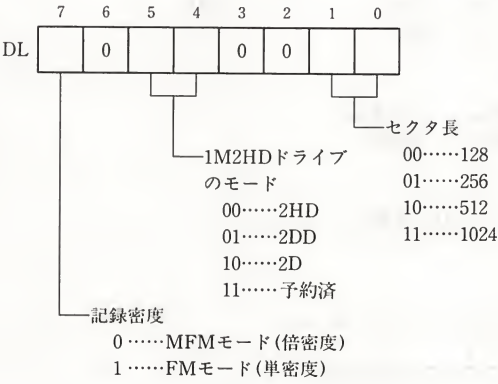
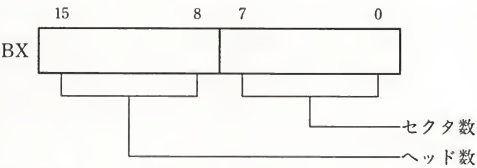
ディスクBIOSについて個別に詳しく解説します.

ディスク	INT 93H
ドライブモードの設定(フロッピディスク)	機能コード00H

エントリ　AH　=00H
AL　=デバイス番号
DL　=ドライブモード1
BX　=ドライブモード2

リターン　AH　=00H(正常終了)
02H(デバイス番号の誤り, ドライブモードの誤り)

説　明　各デバイスのセクタ長, ドライブモード, 記録密度, セクタ数, ヘッド数などを設定します.

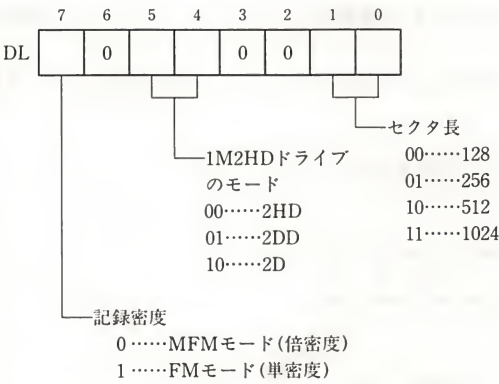
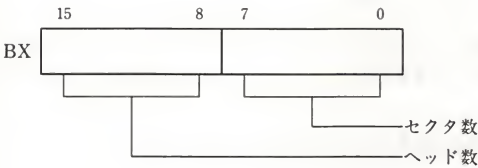
ドライブモード1の形式を示します.

ドライブモード2の形式を示します.

注意) ヘッド数は, 1あるいは2が有効.
(3以上の指定でのフロッピィディスクの動作は, 保証されない.)

ディスク	INT 93H
ドライブモードの取り出し（フロッピィディスク）	機能コード01H

エントリ　AH　＝01H
AL　＝デバイス番号

リターン　AH　＝00H（正常終了）
02H（デバイス番号の誤り）
DL　＝ドライブモード1
BX　＝ドライブモード2

説　明　現在，設定されているセクタ長，ドライブモード，記録密度，セクタ数，ヘッド数の取り出しを行います．

ドライブモード1の形式を示します．

ドライブモード2の形式を示します．

ディスク	INT 93H
ドライブステータス情報の取り出し	機能コード02H

エントリ

AH	＝02H
AL	＝デバイス番号
CH	＝00H

リターン

AH	＝00H（正常終了） 02H（デバイス番号の誤り） 80H（ハードエラー）
DL	＝ドライブステータス（フロッピィディスクのみ）
AL	＝ドライブモード（ハードディスクのみ）
BX	＝最大論理ブロック数（High）（ハードディスクのみ）
DX	＝最大論理ブロック数（Low）（ハードディスクのみ）
CX	＝エラー情報（AHが80Hの場合）

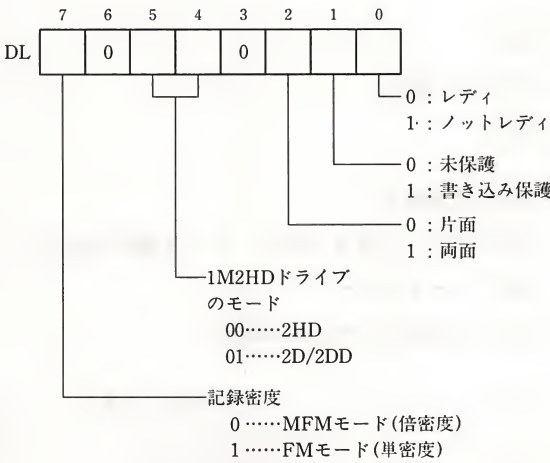
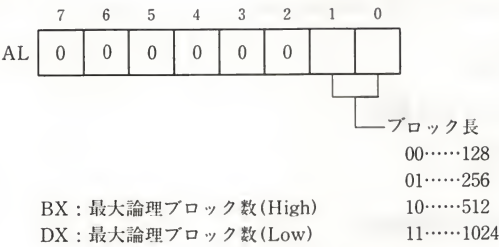
説　明

現在のドライブ状態を取り出します．

1MBドライブでは，ドライブのヘッドをリストアした後，ドライブステータスを返し，正常終了のときは取り出した情報のモードに設定されますが，エラー終了および，1M2HDドライブモードが2D/2DDのときは「ドライブモードの設定（機能コード00H）」によりモードの再設定を行う（2Dと2DDの区別を行うため）必要があります．

この機能はフォーマット済みのフロッピィディスクのみに動作します．

ハードディスクの場合は指定されたハードディスクのブロック長，最大論理ブロック数を返します．

ドライブステータスの形式を示します．

ドライブモードの形式を示します.

ディスク	INT 93H
シリンダ0へのシーク(フロッピィディスク)	機能コード03H

エントリ　AH　＝03H
　　　　　AL　＝デバイス番号

リターン　AH　＝00H(正常終了)
　　　　　　　　02H(デバイス番号の誤り)
　　　　　　　　80H(ハードエラー)
　　　　　CX　＝エラー情報(AHが80Hの場合)

説　明　フロッピディスクのリセットおよび，ヘッドを0シリンダへ移動します.

ディスク	INT 93H
シリンダ0へのシーク(ハードディスク)	機能コード03H

エントリ　AH　＝03H
　　　　　AL　＝デバイス番号
　　　　　CH　＝00H

リターン　AH　＝00H(正常終了)
　　　　　　　　02H(デバイス番号の誤り，ヘッド番号の誤り)
　　　　　　　　80H(ハードエラー)
　　　　　CX　＝エラー情報(AHが80Hの場合)

説　明　ハードディスクのヘッドを0シリンダへ移動します.

ディスク	INT 93H
シーク(フロッピィディスク)	機能コード04H

エントリ

エントリ	AH	=04H
	AL	=デバイス番号
	CX	=シリンダ番号
リターン	AH	=00H(正常終了)
		02H(デバイス番号の誤り)
		80H(ハードエラー)
	CX	=エラー情報(AHが80Hの場合)

説　明　フロッピィディスクの指定されたシリンダアドレスへ，ヘッドを移動します.

ディスク	INT 93H
データの読み出し(フロッピィディスク)	機能コード05H

エントリ	AH	=05H
	AL	=デバイス番号
	CX	=シリンダ番号
	DH	=ヘッド番号
	DL	=セクタ番号
	BX	=セクタ数
	DS:DI	=バッファ先頭アドレス
リターン	AH	=00H(正常終了)
		02H(デバイス番号の誤り，ヘッド番号の誤り)
		80H(ハードエラー)
	BX	=残りのセクタ数
	CX	=エラー情報(AHが80Hの場合)

説　明　フロッピィディスクの指定されたセクタのデータを読み出します.

ディスク　INT 93H

データの読み出し(ハードディスク)　機能コード05H

エントリ	AH	＝05H
	AL	＝デバイス番号
	CH	＝00H
	CL	＝論理ブロック番号(High)
	DX	＝論理ブロック番号(Low)
	BX	＝ブロック数
	DS : DI	＝バッファアドレス
リターン	AH	＝00H(正常終了) 02H(デバイス番号の誤り) 80H(ハードエラー)
	BX	＝残りのブロック数
	CX	＝エラー情報(AHが80Hの場合)

説　明　ハードディスクより指定したブロックのデータを読み出します.

ディスク　INT 93H

データの書き込み(フロッピィディスク)　機能コード06H

エントリ	AH	＝06H
	AL	＝デバイス番号
	CX	＝シリンダ番号
	DH	＝ヘッド番号
	DL	＝セクタ番号
	BX	＝セクタ数
	DS : DI	＝バッファアドレス
リターン	AH	＝00H(正常終了) 02H(デバイス番号の誤り，ヘッド番号の誤り) 80H(ハードエラー)
	BX	＝残りのセクタ数
	CX	＝エラー情報(AHが80Hの場合)

説　明　フロッピィディスクのセクタへデータを書き込みます.

ディスク　INT 93H

データの書き込み(ハードディスク)　機能コード06H

エントリ	AH	=06H
	AL	=デバイス番号
	CH	=00H
	CL	=論理ブロック番号(High)
	DX	=論理ブロック番号(Low)
	BX	=ブロック数
	DS:DI	=バッファアドレス
リターン	AH	=00H(正常終了)
		02H(デバイス番号の誤り)
		80H(ハードエラー)
	BX	=残りのブロック数
	CX	=エラー情報(AHが80Hの場合)

説　明　ハードディスクのブロックにデータを書き込みます.

ディスク　INT 93H

セクタの検査(フロッピィディスク)　機能コード07H

エントリ	AH	=07H
	AL	=デバイス番号
	CX	=シリンダ番号
	DH	=ヘッド番号
	DL	=セクタ番号
	BX	=セクタ数
リターン	AH	=00H(正常終了)
		02H(デバイス番号の誤り，ヘッド番号の誤り)
		80H(ハードエラー)
	BX	=残りのセクタ数
	CX	=エラー情報(AHが80Hの場合)

説　明　フロッピィディスクのセクタのデータを検査します.

ディスク	INT 93H
セクタの検査(ハードディスク)	機能コード07H

エントリ

AH	＝07H
AL	＝デバイス番号
CH	＝00H
CL	＝論理ブロック番号(High)
DX	＝論理ブロック番号(Low)
BX	＝ブロック数

リターン

AH	＝00H(正常終了) 02H(デバイス番号の誤り) 80H(ハードエラー)
BX	＝残りのブロック数
CX	＝エラー情報(AHが80Hの場合)

説明

ハードディスクのブロックのデータを検査します．

ディスク	INT 93H
ハードディスクコントローラのリセット(ハードディスク)	機能コード08H

エントリ

AH	＝08H

リターン

AH	＝00H(正常終了) 80H(ハードエラー)
CX	＝エラー情報(AHが80Hの場合)

説明

ハードエラーが発生した場合，ハードディスクコントローラをリセットします．

ディスク	INT 93H
セクタIDの取り出し(フロッピィディスク)	機能コード09H

エントリ
AH =09H
AL =デバイス番号
CX =シリンダ番号
DH =ヘッド番号
DS:DI =バッファ先頭アドレス

リターン
AH =00H(正常終了)
02H(デバイス番号の誤り)
80H(ハードエラー)
CX =エラー情報(AHが80Hの場合)

説 明
指定トラックのセクタIDをバッファに読み出します.
バッファの形式を示します.

(DS:DI)

0	R	B	トラック番号
1	R	B	ヘッド番号
2	R	B	セクタ番号
3	R	B	セクタ長
4	R	B	CRC
5	R	B	CRC

セクタ長:
0 : 128バイト
1 : 256バイト
2 : 512バイト
3 : 1024バイト

ディスク	INT 93H
トラックのフォーマット(フロッピィディスク)	機能コード0AH

エントリ

AH　=0AH
AL　=デバイス番号
CX　=シリンダ番号
DH　=ヘッド番号
DS:DI　=フォーマットデータ先頭アドレス

リターン

AH　=00H(正常終了)
　02H(デバイス番号の誤り，ヘッド番号の誤り)
　80H(ハードエラー)
CX　=エラー情報(AHが80Hの場合)

説　明

1トラック分のフォーマットを行います．

フォーマットデータの形式は，「第1部第7章　各種のデバイス」を参照してください．

この機能の実行時には，書き込むデータが正しく媒体に書かれたかどうか(ハード上)検出することができないため(リターンが正常であっても，再度読むことができない場合がある)，この機能を実行後に書き込んだ1トラック分のセクタをベリファイ(「セクタの検査(機能コード07H)」)することを推奨します．

ディスク	INT 93H
詳細エラー情報の取り出し(ハードディスク)	機能コード0DH

エントリ

AH　=0DH
AL　=デバイス番号
CH　=00H
DS:DI　=出力エラーアドレス

リターン

AH　=00H(正常終了)
　02H(デバイス番号の誤り)
　80H(ハードエラー)
CX　=エラー情報(AHが80Hの場合)

説　明

詳細なエラー情報を読み出します．

第10章

プリンタBIOS

この章では，プリンタを制御するBIOSについて解説します．

10.1 プリンタBIOS一覧

表II-10-1にプリンタBIOSの一覧を示します．

▼表II-10-1 プリンタBIOS一覧

機能名称	機能コード
プリンタ状態の読み取り	00H
1文字出力	01H
文字列出力	02H

出力形式には，1文字出力と文字列出力があります．1文字出力は印字データ，文字列出力は文字長と文字格納アドレスを指定することによってプリンタに印字できます．

なお，プリンタBIOSは，標準実装セントロニクスインタフェースに接続したプリンタのみをサポートしているので，それ以外のプリンタ番号を指定するとエラーになります．

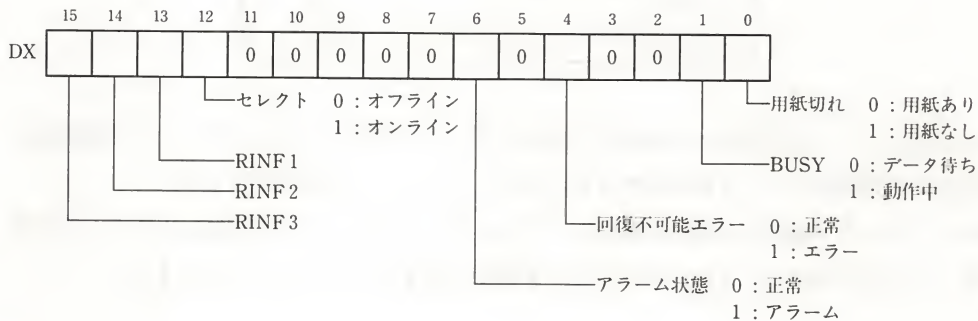
10.2 プリンタBIOSリファレンス

プリンタ BIOS について個別に詳しく解説します.

プリンタ　INT 94H
プリンタ状態の読み取り　機能コード00H

エントリ　AH　＝00H
　　　　　AL　＝00H（標準プリンタポート）

リターン　AH　＝00H（正常終了）
　　　　　　　＝02H（プリンタポートの指定エラー）
　　　　　　　＝03H（標準プリンタポート以外のポートを指定）（FM16β でサポート可であったポート）
　　　　　DX　＝プリンタステータス

説　明　現在のプリンタの情報を通知します.
プリンタステータスの形式を示します.

RINF については，各プリンタの仕様書を参照してください．また，セレクトが1のとき，エラー状態，または，プリンタ情報が返ります(ただし，RINF はプリンタによってはサポートされないことがあります).

アラームビットは，オフライン，用紙切れでオンになり，一度読むとオフになります.

プリンタ	INT 94H
1文字出力	機能コード01H

エントリ　AH　＝01H
　AL　＝00H（標準プリンタポート）
　DL　＝印字データのキャラクタコード

リターン　AH　＝00H（正常終了）
　02H（プリンタポートの指定エラー）
　03H（標準プリンタポート以外のボードを指定－FM16βでサポート可であったポート）
　04H（プリンタの用紙切れ）
　05H（プリンタがノットレディ状態）
　80H（プリンタに動作エラーが発生した）

説　明　プリンタに印字データを送ります．
文字を印字するときは，その文字コードを送った後にプリンタのLF，CRコードも送らない限り，印字されません．

プリンタ	INT 94H
文字列出力	**機能コード02H**

エントリ	AH	＝02H
	AL	＝00H（標準プリンタポート）
	CX	＝データ長（バイト数）
	DS : DI	＝文字列データアドレス
リターン	AH	＝00H（正常終了） 02H（プリンタポートの指定エラー） 03H（標準プリンタポート以外のボードを指定－FM16βでサポート可であったポート） 04H（プリンタの用紙切れ） 05H（プリンタがノットレディ状態） 80H（プリンタに動作エラーが発生した）
	CX	＝残り文字のバイト数

説　明

プリンタへ文字列を送ります．

印字中にプリンタのエラーが発生したときには，印字されなかった残りのバイト数を通知します．プリンタ内にあるバッファに残っているデータの印字については保証されません．

第 11 章

時計をサポートするBIOS

この章では，時計をサポートする BIOS について解説します．
時計サポートの BIOS としてはカレンダ時計，タイマ管理，時計管理の 3 種類があります．

11.1 時計をサポートする BIOS 一覧

表II-11-1に，カレンダ時計，タイマ管理，時計管理の BIOS の一覧を示します．

▼表II-11-1　時計をサポートする BIOS 一覧

カレンダ時計 BIOS

機能名称	機能コード
日付/時刻の設定	00H
日付/時刻の読み取り	01H

タイマ管理 BIOS

機能名称	機能コード
タイマの登録	00H
タイマの取り消し	01H
タイマのカウント値の読み取り	02H

時計管理 BIOS

機能名称	機能コード
指定時刻の割り込み処理の登録	00H
指定時刻の割り込み処理の取り消し	01H

1．カレンダ時計 BIOS

システムのカレンダ時計に対して，日付／時刻の設定と読み取りを行います．

2．タイマ管理 BIOS

指定された周期で割り込みを発生し，その解除を行うことができます．

このBIOSを利用することにより，通常の処理とは別に一定時間間隔で作業を行うことができます．例えば，一定時間間隔で状態変化を見たりするときに有効です．

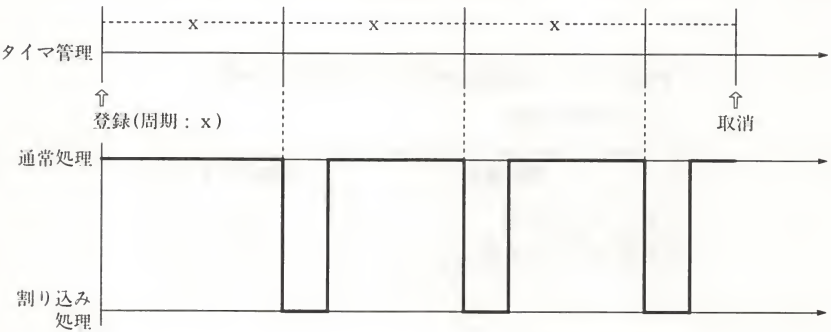
割り込みはタイマの登録が正常に行われた時点から有効になります．

割り込み先では，必要な処理の後，必ずタイマ管理に制御をもどしてください．すなわち，リターンすることが必要です．このとき，レジスタ値は保存されていなくても支障ありません．

割り込み周期の最小単位は，10msです．

タイマ割り込みルーチンは，他へ与える影響(システムの時間的負荷)が大きいので，割り込み処理内では，フラグを立てる程度の処理にしてください．

図II-11-1にタイマ管理BIOSの割り込み周期を示します．

▼図II-11-1　タイマ管理BIOSの割り込み周期

3．時計管理 BIOS

指定時刻での割り込みの発生と，その解除を行います．制御の流れは，タイマと同じです．

11.2　時計をサポートするBIOSリファレンス

時計をサポートするBIOSについて個別に解説します．

カレンダ時計　INT 96H

日付／時刻の設定　機能コード00H

エントリ　AH　＝00H
DS : DI　＝日付／時刻パラメータブロックアドレス

リターン　AH　＝00H（正常終了）
02H（日付／時刻の指定エラー）

説　明　指定された日付／時刻を，カレンダ時計に設定します．
日付／時刻の指定は，日付／時刻パラメータブロックで指定します．
日付／時刻パラメータブロックの形式を示します．

(DS : DI)

0	E	W	年(1980～2079)
2	E	B	月(1～12)
3	E	B	日(1～31)
4	E	B	0
5	E	B	時(0～23)
6	E	B	分(0～59)
7	E	B	秒(0～59)
8	E	B	0
9	E	B	0

各パラメータ値は，バイナリで，括弧内の範囲に指定してください．それ以外の値を指定した場合は，エラーとなります．

曜日は内部で自動設定されます．

カレンダ時計　INT 96H

日付／時刻の読み取り　機能コード01H

エントリ　AH　＝01H

DS : DI　＝日付／時刻パラメータブロックアドレス

リターン　AH　＝00H（正常終了）

説　明　カレンダ時計から現在の日付／時刻を取得します．

取得した日付／時刻は，パラメータブロックにセットされます．

日付／時刻パラメータブロックの形式を示します．

(DS : DI)

0	R	W	年(1980～2079)
2	R	B	月(1～12)
3	R	B	日(1～31)
4	R	.B	曜(0～6)
5	R	B	時(0～23)
6	R	B	分(0～59)
7	R	B	秒(0～59)
8	R	B	1/100秒(0～99)
9	R	B	0

各パラメータ値は，バイナリです．

曜日はそれぞれ，次のようになります．0＝日，1＝月，2＝火，3＝水，4＝木，5＝金，6＝土．

タイマ管理	INT 97H
タイマの登録	機能コード00H

エントリ　AH　＝00H
DS:DI　＝タイマ管理パラメータブロックアドレス

リターン　AH　＝00H (正常終了)
02H (入力パラメータエラー)
03H (登録に失敗した)
AL　＝タイマ管理番号

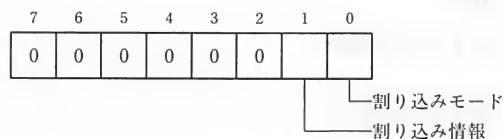
説　明　タイマ割り込み処理の登録をします．
タイマ割り込みの詳細は，タイマ管理パラメータブロックで指定します．
タイマ管理パラメータブロックの形式を示します．

(DS : DI)

0	E	B	タイマ割り込みモード
1	E	B	0
2	E	D	タイマ割り込み処理アドレス
6	E	W	タイマ周期(HIGH)
8	E	W	タイマ周期(LOW)

それぞれのパラメータの詳細を説明します．

●タイマ割り込みモード

次の2つのモードがあります．

割り込みモード

0：インターバルモード(指定時間間隔で割り込み処理を起動し，取り消されない限り同じ時間間隔で，割り込み処理をし続ける)．

1：ワンショットモード(指定時間経過後，割り込み処理を起動する．割り込み処理は1回のみで，割り込み処理後，そのタイマはタイマ管理により自動的に取り消されるので，タイマの取り消しをする必要はない)．

割り込み情報

0：割り込みモード(指定された周期で指定アドレスに割り込みを行う).

1：カウントモード(タイマカウントのみ行うことの指定であり,割り込み処理を起動することはしない).

●タイマ割り込み処理アドレス

タイマ割り込み処理のオフセットアドレスとセグメントアドレス(カウントモードの場合，ALL0).

●タイマ周期(HIGH/LOW)

タイマの周期を示します.

周期の上位2バイトをHIGHに，下位2バイトをLOWに設定してください．周期は10ms単位に指定します(1＝10ms，2＝20ms，3＝30ms,..........).

周期として0を指定した場合はパラメータエラーとなります.

タイマ割り込みが発生した場合，指定されたタイマ割り込み処理にセグメント間コール(far　CALL)します．処理を終了したときはセグメント間リターン(far RETURN)を行ってください.

割り込み処理内でのレジスタの保護は，必要ありません.

タイマ管理	INT 97H
タイマの取り消し	機能コード01H

エントリ　AH　＝01H
　　　　　AL　＝タイマ管理番号

リターン　AH　＝00H(正常終了)
　　　　　　　02H(タイマ管理番号エラー)

説　明　指定されたタイマ管理番号のタイマ割り込み処理を取り消します.
タイマ管理番号は，登録済のタイマ管理番号でなければなりません.

タイマ管理	INT 97H
タイマのカウント値の読み取り	機能コード02H

エントリ　AH　＝02H
AL　＝タイマ管理番号
DS：DI　＝タイマ管理パラメータブロックアドレス

リターン　AH　＝00H（正常終了）
02H（タイマ管理番号エラー）

説　明

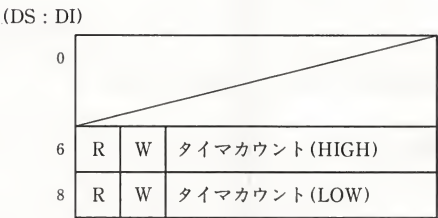
タイマ管理では，登録されているタイマごとに10ms単位でカウントを行っています．

このカウント値は，タイマの登録時に指定された周期に達すると0にもどされ，再びカウントを始めます（例えば周期が100（1000ms）の場合，タイマカウント値は0～99の値になります）．

この機能は，指定されたタイマ管理番号の，取得要求があった時点でのタイマカウント値を返すものです．

取得したタイマカウント値は，タイマ管理パラメータブロックに格納されます．

タイマ管理番号は，登録済のタイマ管理番号でなければなりません．

タイマ管理パラメータブロックの形式を示します．

●タイマカウント（HIGH/LOW）

取得要求があった時点のタイマカウント値を示します．

カウント値の上位2バイトがHIGHに，下位2バイトがLOWに設定されます．

時計管理　INT 98H

指定時刻の割り込み処理の登録　機能コード00H

エントリ
AH　＝00H
DS:DI　＝時計管理パラメータブロックアドレス

リターン
AH　＝00H（正常終了）
　02H（時計管理パラメータブロックの指定エラー）
　03H（登録に失敗した）
AL　＝時計管理番号

説　明
指定時刻の割り込み処理を登録します．

登録が正常に行われると，時計管理パラメータブロックにより指定された年，月，日，時，分，秒から，割り込みを開始します．割り込みが発生した場合，指定された割り込み処理にセグメント間コール(far CALL)します．処理を終了したときにはセグメント間リターン(far RETURN)を行ってください．割り込み処理内でのレジスタの保護は，必要ありません．

時計管理パラメータブロックは，割り込み制御部分と，時刻指定部分の2つに分かれています．形式を次に示します．

(DS : DI)

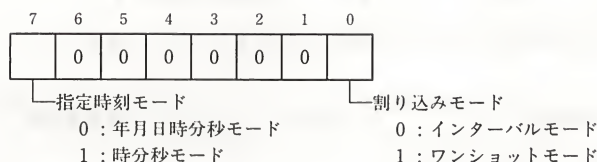
0	E	B	設定フラグ	割り込み制御
1	E	B	0	
2	E	D	時計管理割り込み処理アドレス	
6	E	W	割り込み周期	
8	E	W	0	
A	E	W	年(1980～2079)	時刻指定
C	E	B	月(1～12)	
D	E	B	日(1～31)	
E	E	B	0	
F	E	B	時(0～23)	
10	E	B	分(0～59)	
11	E	B	秒(0～59)	

時刻指定パラメータは，バイナリ形式で括弧内の値の範囲で指定してください．それ以外の値を指定した場合は，エラーとなります．

曜日は内部で自動設定されます．

各パラメータの形式について説明します．

●設定フラグ

次の2つの情報があります．

割り込みモード

インターバルモード：指定時刻から指定時間間隔で割り込み処理を起動し，取り消さない限り指定周期で割り込み処理をし続ける．

ワンショットモード：指定時刻を通過すると，1回だけ割り込み処理を起動する(割り込み周期は意味を持たない)．

指定時刻モード

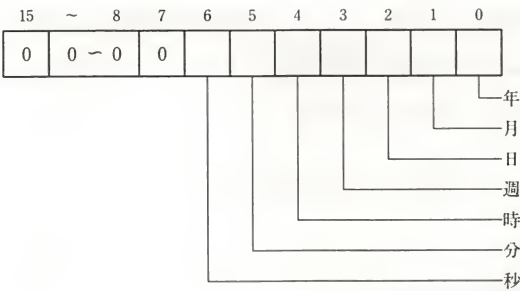
年月日時分秒モード：指定されたパラメータブロックの年月日時分秒から，割り込みを開始する．

時分秒モード　　　：パラメータブロックの年月日部分の指定を無効とし(指定された時分秒は有効)，このオペレーションが実行された時点での年月日が指定されたものとする．すなわち，当日の今後の割り込みに対して用いられる．

●時計管理割り込み処理アドレス

時計管理割り込み処理のオフセットアドレスとセグメントアドレスを指定します．

●割り込み周期

割り込み時刻の間隔を指定します．

各ビットがON(1)のとき，その周期が有効です．
割り込み周期と割り込み時刻の関係を次に示します．

指定時刻		1985年10月4日(金)　11時8分30秒						
割り込み周期ビット	年	毎年	↑	↑	↑	↑	↑	↑
	月		毎月	↑	↑	↑	↑	↑
	日			毎日	↑	↑	↑	↑
	週				毎週	↑	↑	↑
	時					毎時	↑	↑
	分						毎分	↑
	秒							毎秒

指定時刻から，割り込みが開始します．
例えば，割り込み周期が年の場合，指定時刻から毎年同じ月日の同じ時刻に，割り込みが発生します．
なお，割り込み周期は，複数指定できません．必ず1つのビットのみがONで，その他はOFFにしてください．複数指定した場合はパラメータエラーとなります．

時計管理　INT 98H
指定時刻の割り込み処理の取り消し　機能コード01H

エントリ　AH　＝01H
AL　＝時計管理番号

リターン　AH　＝00H(正常終了)
02H(時計管理番号エラー)

説　明　指定された時計管理番号の，指定時刻の割り込み処理を取り消します．

第12章

RS-232C BIOS

この章では，RS-232Cの制御を行うBIOSについて解説します．

FM TOWNSには，RS-232Cポートとして，本体に実装されているRS-232Cインタフェースと，オプションの内蔵モデムの2系統があります．

RS-232C BIOSは，両方の制御が可能です．

12.1 RS-232C BIOS一覧

表II-12-1に，RS-232C BIOSの一覧を示します．

▼表II-12-1　RS-232C BIOS一覧

機能名称	機能コード
シリアルポートの検出	00H
回線オープン	01H
回線クローズ	02H
通信パラメータの設定	03H
通信パラメータの読み取り	04H
受信バッファ内部有効データ数の読み取り	05H
データの受信	06H
データの送信	07H
シリアルポートの制御	08H
ステータス情報の読み取り	09H
受信バッファの初期化	0AH
ブレーク信号の送信	0BH
拡張割り込みの設定	0CH
拡張割り込みの読み取り	0DH
拡張DTR信号の保持設定	0EH
XOFF受信のクリア	0FH
送信バッファ内有効データ数の読み取り	10H

内蔵モデムを接続しない状態では，本体に実装されているRS-232Cインタフェースが有効で，内蔵モデムを接続すると内蔵モデムが有効となります．

なお，BIOSにおいてポートの指定をする際には，どちらの場合もポート0を指定してください．

12.2　RS-232C BIOS リファリンス

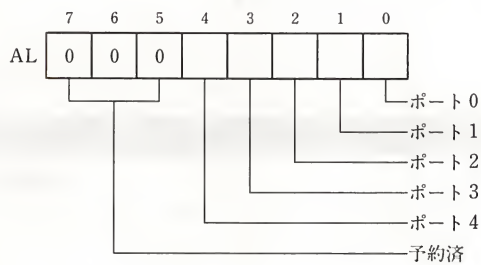
RS-232C BIOS について個別に詳しく解説します．

RS-232C	INT 9BH
シリアルポートの検出	機能コード00H

エントリ　AH　＝00H

リターン　AH　＝00H（正常終了）
　　　　　AL　＝検出データ

説　明　拡張 I/O ユニット内に入るインタフェースカードの接続状態を読み取ります．

検出データは次のようになります．

ビット対応により，0：非接続，1：接続となります．

ポート0は内蔵のため1となります．

RS-232C	INT 9BH
回線オープン	**機能コード01H**

エントリ
AH　=01H
AL　=ポート番号(0～4)

リターン
AH　=00H(正常終了)
02H(範囲以外のポート番号を指定した)
03H(拡張カードが接続されていないポートを指定した)
05H(回線がクローズ状態でない)
07H(通信パラメータが設定されていないのに回線をオープンしようとした)

説　明
「通信パラメータの設定(機能コード03H)」で設定したモードでオープンを行います.

通信パラメータを設定しない場合，または，すでにオープンされている状態ではオープンできません．回線オープン時には受信バッファ内の文字カウンタ，入力ポインタ，出力ポインタを初期化します.

RS-232C	INT 9BH
回線クローズ	**機能コード02H**

エントリ
AH　=02H
AL　=ポート番号(0～4)

リターン
AH　=00H(正常終了)
02H(範囲以外のポート番号を指定した)
03H(拡張カードが接続されていないポートを指定した)
04H(回線がオープン状態でない)

説　明
回線のクローズ(回線による割り込みの禁止)を行います.

ポートが送信バッファに指定されている場合，送信バッファ内に残っている出力待ちデータはすべて無効になります.

注意)　送信バッファ指定されている場合に,このコマンドを使用するときは,事前に送信バッファ内の有効データ読み取り機能を利用し，データ数=0を確認後クローズしてください.

RS-232C INT 9BH

通信パラメータの設定　機能コード03H

エントリ　AH　＝03H
AL　＝ポート番号(0～4)
DS:DI　＝通信パラメータアドレス

リターン　AH　＝00H(正常終了)
02H(範囲以外のポート番号を指定した)
03H(拡張カードが接続されていないポートを指定した)
05H(回線がクローズ状態でない)
06H(受信バッファ領域が確保されていない)

説　明　シリアルポートの通信パラメータを設定します.

オープン状態でパラメータを変更することはできません. 通信パラメータを設定すると「通信パラメータの読み取り(機能コード04H)」で,『通信パラメータが設定されていないのに・・』のエラーは発生しません.

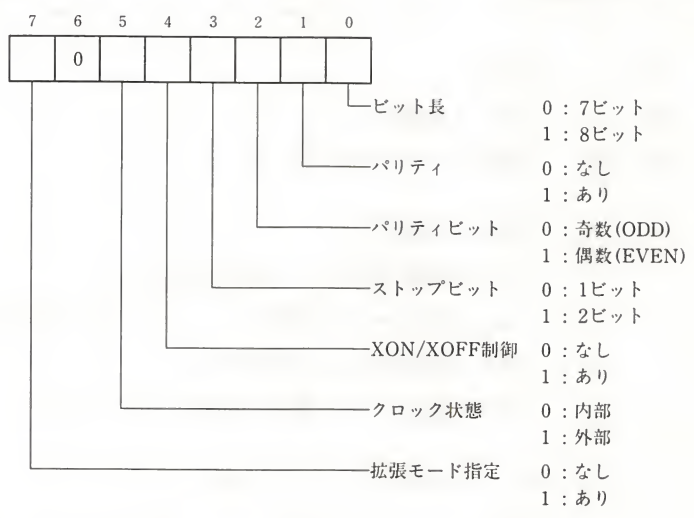
通信パラメータの形式を示します.

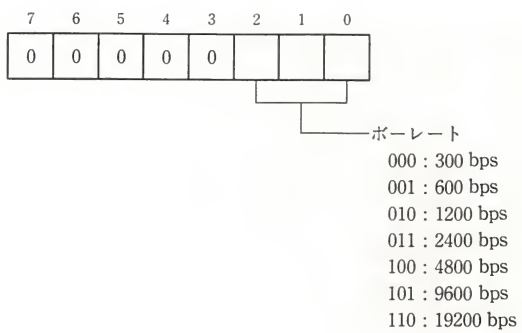
(DS:DI)

0	E	B	通信モード
1	E	B	ボーレート
2	E	D	受信バッファアドレス
6	E	W	送信タイムアウト時間
8	E	W	受信タイムアウト時間
A	E	D	受信通知アドレス
E	E	B	拡張通信モード
F	E	B	XON コード
10	E	B	XOFF コード
11	E	D	送信バッファアドレス

注意) オフセットアドレス0EH以降は通信モードの拡張モード指定(ビット7)がONのときしか設定できないので, OFFのときは設定する必要はありません.

各パラメータを説明します.

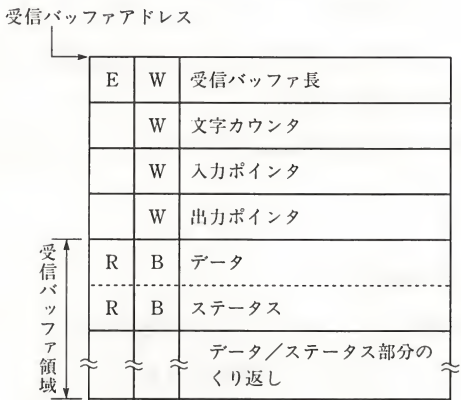
●通信モード

●ボーレート

シリアルポートのボーレートを設定します.

●受信バッファ

シリアルポートから受信したデータを格納するバッファを指定します.

受信バッファアドレス

E	W	受信バッファ長
	W	文字カウンタ
	W	入力ポインタ
	W	出力ポインタ
R	B	データ
R	B	ステータス
		データ/ステータス部分のくり返し

(受信バッファ領域: データ/ステータス部分)

バッファ長　：受信バッファ領域の大きさを示す
文字カウンタ：受信バッファ内に入っているデータ数を示す
入力ポインタ：次に入力される文字のバッファ位置を示す
出力ポインタ：次に出力される文字のバッファ位置を示す

注意1) 受信バッファ内でのセグメントオーバーランを防ぐため，受信バッファアドレス(オフセット)を0にするか，受信バッファ領域を受信バッファ長より16バイト余分に確保してください.

注意2) 文字カウンタ，入力ポインタ，出力ポインタは回線オープン時に初期化を行います.

●タイムアウト時間

送信/受信のタイムアウト時間を設定します.

指定時間 (1～65534×10ms)	意 味
0	送/受信の結果にかかわらず即時復帰を行う.
1～65534	指定時間内に送/受信が完了しない場合はタイムアウトとなり強制復帰を行う.
65535	送/受信が完了するまで復帰しない.

●受信通知アドレス

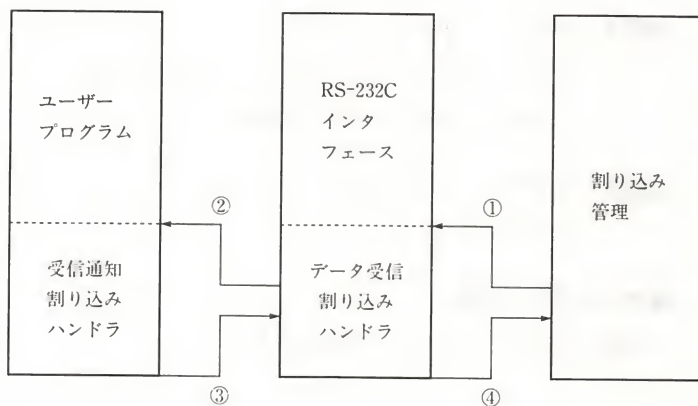
回線からデータを受信したことを通知する，処理ルーチンのアドレスを設定します．データを受信した場合，処理ルーチンへセグメント間コール(far CALL)するので，処理が終了したときは，セグメント間リターン(far RETURN)を行ってください．処理ルーチンに制御が渡ったとき，ルーチン内でのレジスタの保護は必要ありません．

また，通知する場合は受信通知ステータスをいっしょに渡します．

受信通知ステータス

AH ＝00H(正常終了)
　　＝10H(受信バッファ内のデータがあふれた)
　　＝11H(ストップビットが検出できない)
　　＝12H(正しいパリティビットが検出できない)
AL ＝受信ポート番号

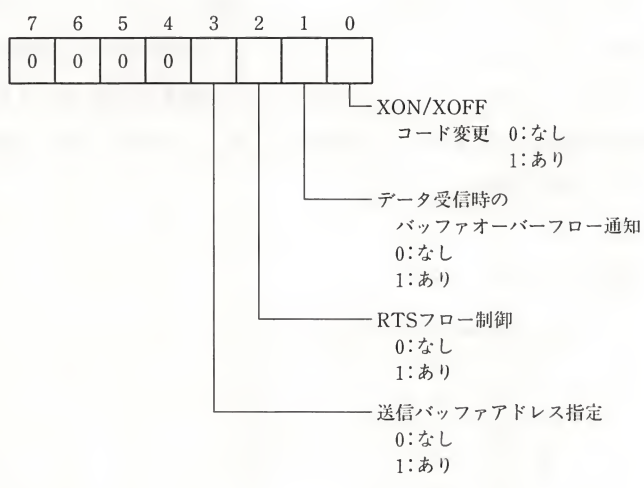
注意) セグメントとオフセットアドレスの値が0のときは通知しません．

受信通知割り込み処理の流れ

①回線よりデータが受信され，割り込み管理からRS-232Cの割り込みハンドラへ制御が渡る．

②データを受信バッファに入れ，ユーザーの割り込みハンドラが登録されている場合にそのハンドラへ通知を行う．

③受信通知ハンドラからRS-232Cの割り込みハンドラへ復帰を行う

④ RS-232Cの割り込みハンドラから割り込み処理への復帰を行う．

●拡張通信モード

● RTS フロー制御

XON/XOFF のバッファ制御の代わりに，RTS 信号の ON/OFF でフロー制御を行います．

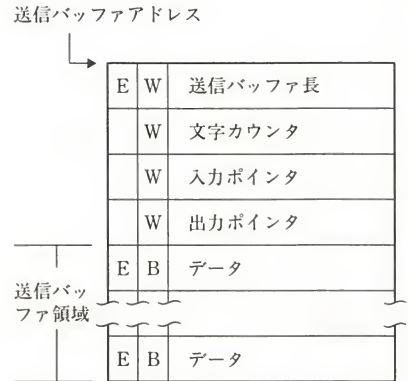
データ受信時に，受信バッファがいっぱいになると RTS 信号を OFF にし，受信バッファが受信可能になると RTS 信号を ON にします．

このビットを 1 に設定すると，「通信モード」の XON/XOFF 制御の指定が無視され，RTS フロー制御になります．

●送信バッファアドレス指定

割り込みを使用してデータの送信を行います．割り込みで送信を行うため，データ送信中に他の処理を行うことが可能です．

●送信バッファ

シリアルポートに送信するデータを格納しておくバッファを指定します．送信バッファ内でのセグメントオーバーランを防ぐため，送信バッファアドレス(オフセット)を0にするか，送信バッファ領域を送信バッファ長より16バイト余分に確保してください．文字カウンタ，入力ポインタ，出力ポインタは回線オープン時に初期化されます．

バッファ長　：送信バッファ領域の大きさを示します
文字カウンタ：送信バッファ内に入っているデータ数を示します
入力ポインタ：次に入力される文字のバッファ位置を示します
出力ポインタ：次に出力される文字のバッファ位置を示します

● XON/XOFFコード

XON/XOFFのバッファ制御コードを指定します．

このコードは拡張通信モードのXON/XOFFコード変更(ビット0)が1の場合にのみ設定されます．

RS-232C	INT 9BH
通信パラメータの読み取り	機能コード04H

エントリ　AH　＝04H
AL　＝ポート番号(0～4)
DS:DI　＝通信パラメータアドレス

リターン　AH　＝00H(正常終了)
02H(範囲以外のポート番号を指定した)
03H(拡張カードが接続されていないポートを指定した)
07H(通信パラメータが設定されていないのに通信パラメータを取得しようとした)

説　明　「通信パラメータの設定(機能コード03H)」で設定した通信パラメータを指定された通信パラメータ領域に読み込みます.

通信パラメータの形式を示します.

(DS:DI)

0	E	B	通信モード
1	E	B	ボーレート
2	E	D	受信バッファアドレス
6	E	W	送信タイムアウト時間
8	E	W	受信タイムアウト時間
A	E	D	受信通知アドレス
E	E	B	拡張通信モード
F	E	B	XON コード
10	E	B	XOFF コード
11	E	D	送信バッファアドレス

注意1) 受信バッファアドレスはセグメントオーバーラン防止のため,「通信パラメータの設定(機能コード03H)」で設定した値と異なる場合があります.

注意2) オフセットアドレス0EH以降は通信モードの拡張モード指定(ビット7)がONのときのみ読み取りができ, OFFのときは読み取りができません.

RS-232C　　INT 9BH

受信バッファ内有効データ数の読み取り　　機能コード05H

エントリ	AH	＝05H
	AL	＝ポート番号(0～4)
リターン	AH	＝00H(正常終了) 02H(範囲以外のポート番号を指定した) 03H(拡張カードが接続されていないポートを指定した) 04H(回線がオープン状態でない)
	DX	＝データ数

説　明　　受信バッファ内に入っているデータ数を返します.

RS-232C	INT 9BH
データの受信	機能コード06H

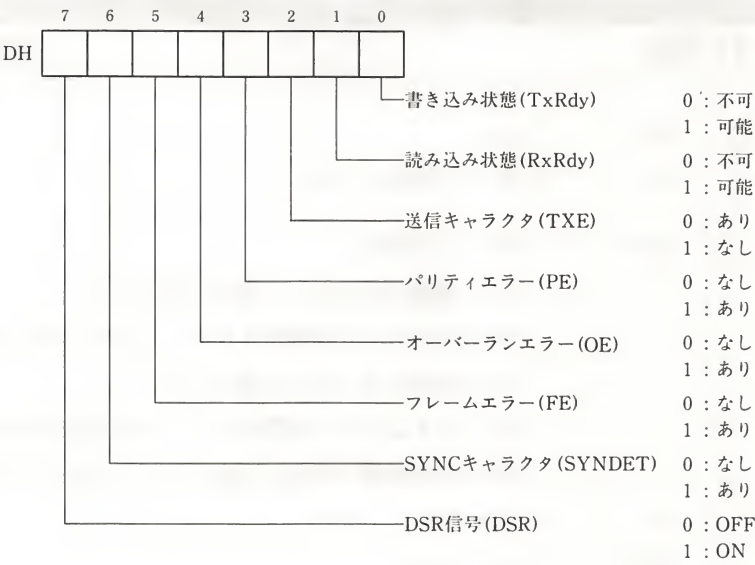
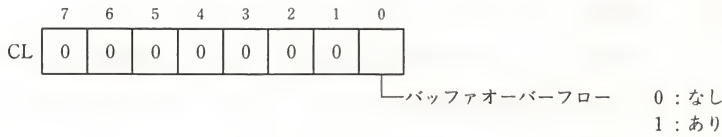
エントリ　AH　=06H
　　　　　AL　=ポート番号(0～4)

リターン　AH　=00H(正常終了)
02H(範囲以外のポート番号を指定した)
03H(拡張カードが接続されていないポートを指定した)
04H(回線がオープン状態でない)
08H(タイムアウト時間内にデータの受信ができなかった)
0AH(即時復帰の場合に受信バッファにデータがなかった)
　　　　　DL　=データ
　　　　　DH　=ステータス
　　　　　CL　=拡張ステータス
　　　　　CH　=00H

説　明　受信バッファ内に入っているデータを受け渡します。

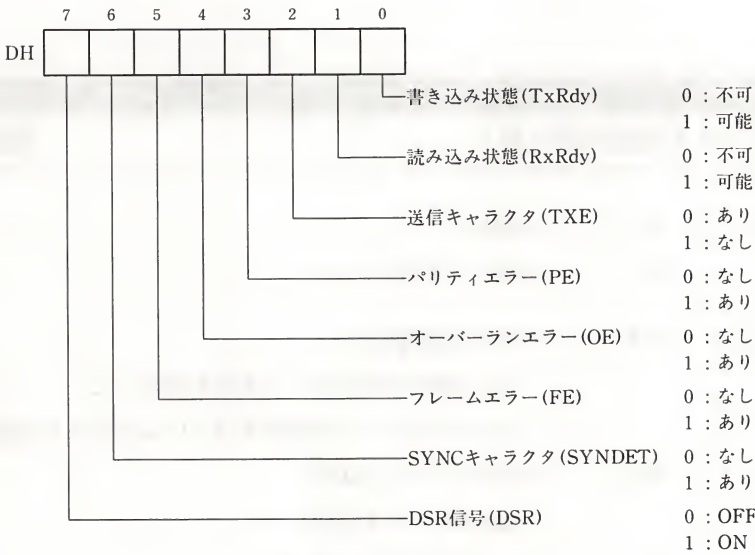
受信バッファが空の場合，受信タイムアウト時間内にデータの受信ができなかったときは強制復帰を行います。XON/XOFF制御ありのときは，XONコードとXOFFコードはデータになりません。バッファオーバーフローになった場合，あふれたデータは捨てられます。

注意）CLとCHは拡張通信モードの「データ受信時のバッファオーバーフローの通知」が1のときのみ有効です。「バッファオーバーフローの通知」が0のときはCLとCHの値は保存されます。

ステータスの形式を示します。

拡張ステータスの形式を示します．

RS-232C	INT 9BH
データの送信	機能コード07H

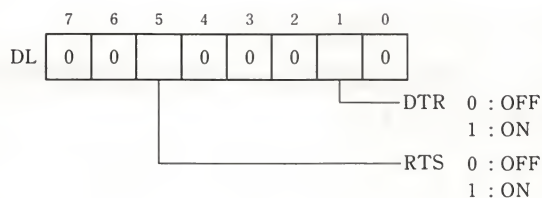
エントリ　AH　＝07H
　　　　　AL　＝ポート番号（0～4）
　　　　　DL　＝データ

リターン　AH　＝00H（正常終了）
　　　　　　　02H（範囲以外のポート番号を指定した）
　　　　　　　03H（拡張カードが接続されていないポートを指定した）
　　　　　　　04H（回線がオープン状態でない）
　　　　　　　08H（タイムアウト時間内にデータの送信ができなかった）
　　　　　　　09H（XOFF コードを受信していたため，タイムアウト時間内にデータの送信ができなかった）
　　　　　　　0BH（即時復帰の場合にデータの送信できる状態でなかった）
　　　　　DH　＝ステータス

説　明　シリアルポートにデータを書き込みます．
送信タイムアウト時間内に送信できない場合には強制復帰を行います．
ステータスの形式を示します．

RS-232C	INT 9BH
シリアルポートの制御	機能コード08H

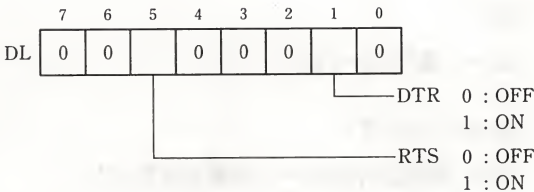
エントリ　AH　＝08H
　　　　　AL　＝ポート番号（0～4）
　　　　　DL　＝制御データ

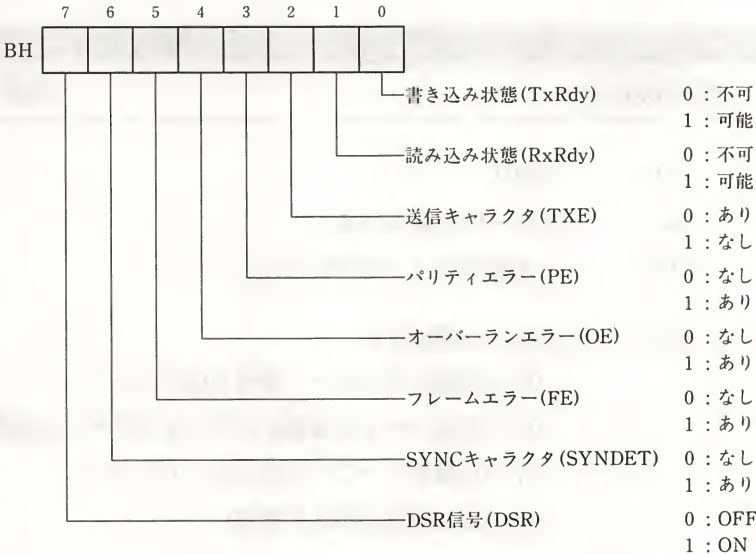
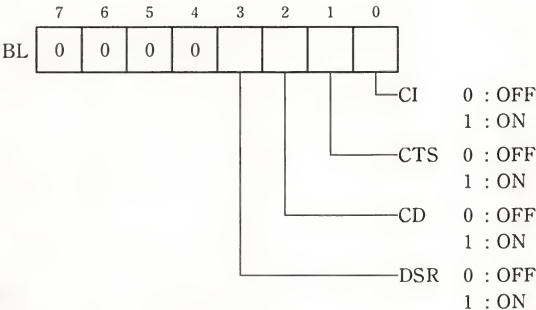
リターン　AH　＝00H（正常終了）
　　　　　　　　02H（範囲以外のポート番号を指定した）
　　　　　　　　03H（拡張カードが接続されていないポートを指定した）
　　　　　　　　04H（回線がオープン状態になっていない）

説　明　シリアルポートのDTR/RTSの信号をコントロールします．
このモードは，次にこの機能が呼び出されて書き換えられるか，回線がクローズされるまで続きます．
制御データの形式を示します．

RS-232C	INT 9BH
ステータス情報の読み取り	機能コード09H

エントリ　AH　＝09H
　　　　　AL　＝ポート番号（0～4）

リターン　AH　＝00H（正常終了）
　　　　　　　　02H（範囲以外のポート番号を指定した）
　　　　　　　　03H（拡張カードが接続されていないポートを指定した）
　　　　　DH　＝00H（オープン状態）
　　　　　　　　01H（クローズ状態）
　　　　　DL　＝シリアルポートの状態
　　　　　BH　＝ステータス
　　　　　BL　＝信号線状態

説　明　シリアルポートのステータス情報を通知します。
シリアルポートの状態を示します。

これは，「シリアルポートの制御(機能コード08H)」で設定したDTR/RTS信号の状態です。

ステータスの形式を示します。

信号線状態の形式を示します。

RS-232C　INT 9BH
受信バッファの初期化　機能コード0AH

エントリ	AH	＝0AH
	AL	＝ポート番号（0～4）
リターン	AH	＝00H（正常終了）
		02H（範囲以外のポート番号を指定した）
		03H（拡張カードが接続されていないポートを指定した）
		04H（回線がオープン状態になっていない）

説　明　受信バッファ内のデータカウント，各ポインタを初期化します．

RS-232C　INT 9BH
ブレーク信号の送出　機能コード0BH

エントリ	AH	＝0BH
	AL	＝ポート番号（0～4）
	DX	＝送信時間（1～65535×10ms）
リターン	AH	＝00H（正常終了）
		02H（範囲以外のポート番号を指定した）
		03H（拡張カードが接続されていないポートを指定した）
		04H（回線がオープン状態になっていない）
		06H（送信時間の指定が異常）

説　明　送信時間が終了するまでブレーク信号を送信します．

送信時間の指定が0のときはエラーとなります．

RS-232C　INT 9BH

拡張割り込みの設定　機能コード0CH

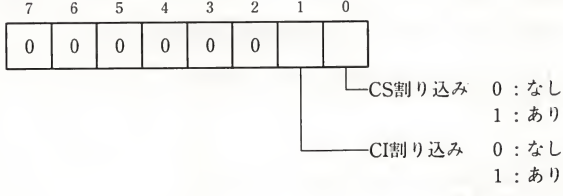
エントリ　AH　＝0CH
　　　　　AL　＝ポート番号（0～4）
　　　　　DS : DI　＝割り込みパラメータアドレス

リターン　AH　＝00H（正常終了）
　　　　　　　　02H（範囲以外のポート番号を指定した）
　　　　　　　　03H（拡張カードが接続されていないポートを指定した）

説　明　拡張割り込みの設定を行います．
割り込みパラメータの形式を示します．

(DS : DI)

0	E	B	割り込みフラグ
1	E	D	CS信号割り込みアドレス
5	E	D	CI信号割り込みアドレス

割り込みフラグの形式を示します．

CS割り込みアドレスには，回線からCS(Clear to Send)による割り込みがあったことを通知する処理ルーチンのアドレスを設定します．

この指定は，割り込みフラグのCS割り込み（ビット1）が1の場合のみに設定されます．なお，セグメントとオフセットの値が0のときは通知しません．

CI割り込みアドレスには，回線からCI(Calling Indicator)による割り込みがあったことを通知する処理ルーチンのアドレスを設定します．

この指定は割り込みフラグのCI割り込み(ビット1)が1の場合のみに設定されます．なお，セグメントとオフセットの値が0のときは通知しません．

注意) CS割り込みとCI割り込みは割り込みが入った場合，処理ルーチンへセグメント間コール(far CALL)しますので，処理が終了したときは，セグメント間リターン(far RETURN)を行ってください．
　処理ルーチンに制御が渡ったとき，ルーチン内でのレジスタの保護は必要ありません．

RS-232C　INT 9BH
拡張割り込みの読み取り　機能コード0DH

エントリ
AH　=0DH
AL　=ポート番号(0～4)
DS:DI　=割り込みパラメータアドレス

リターン
AH　=00H(正常終了)
　02H(範囲以外のポート番号を指定した)
　03H(拡張カードが接続されていないポートを指定した)

説　明
「拡張割り込みの設定(機能コード0CH)」で設定した割り込みパラメータを，指定した割り込みパラメータ用領域に読み出します．
　割り込みパラメータの形式を示します．

(DS : DI)

0	E	B	割り込みフラグ
1	E	D	CS信号割り込みアドレス
5	E	D	CI信号割り込みアドレス

RS-232C	INT 9BH
拡張 DTR 信号の保持設定	機能コード0EH

エントリ　AH　＝0EH
　　　　　AL　＝ポート番号(0～4)
　　　　　DL　＝00H(保持しない)
　　　　　　　　01H(保持する)

リターン　AH　＝00H(正常終了)
　　　　　　　　02H(範囲以外のポート番号を指定した)
　　　　　　　　03H(拡張カードが接続されていないポートを指定した)

説　明　「回線クローズ(機能コード02H)」を行ってもDTR信号がOFFされないように保持します．

なお，この機能は「回線オープン(機能コード01H)」を行うと無効になります．

初期状態では「保持しない」になっています．

RS-232C	INT 9BH
XOFF 受信のクリア	機能コード0FH

エントリ　AH　＝0FH
　　　　　AL　＝ポート番号(0～4)

リターン　AH　＝00H(正常終了)
　　　　　　　　02H(範囲以外のポート番号を指定した)
　　　　　　　　03H(拡張カードが接続されていないポートを指定した)
　　　　　　　　04H(回線がオープン状態でない)

説　明　XON/XOFF制御を行っているときには，XOFFの後には通常XONがきますが，何らかの原因で，XONがこなかった場合は，送信不能となります．このような場合，このファンクションを実行することにより，XOFFの受信がクリアされ，送信可能になります．

RS-232C	INT 9BH
送信バッファ内有効データ数の読み取り	機能コード10H

エントリ　AH　＝10H
　　　　　AL　＝ポート番号(0～4)

リターン　AH　＝00H(正常終了)
　　　　　　　　02H(範囲以外のポート番号を指定した)
　　　　　　　　03H(拡張カードが接続されていないポートを指定した)
　　　　　　　　04H(回線がオープン状態でない)
　　　　　DX　＝データ数

説　明　送信バッファ内に入っているデータ数をDXに返します.

第13章

ブザーBIOS

この章では，本体に内蔵されているブザー(BEEP音)のON/OFFの制御を行うブザーBIOSについて解説します．

13.1 ブザーBIOS一覧

表II-13-1に，ブザーBIOSの一覧を示します．

▼表II-13-1　ブザーBIOS一覧

機能名称	機能コード
ブザーON	00H
ブザーOFF	01H
ブザーON(一定時間)	02H
ブザーON(カウンタ数,指定時間)	03H
ブザー情報の読み取り1	04H
ブザーON(周波数，指定時間)	05H
ブザー情報の読み取り2	06H

13.2 ブザーBIOSリファレンス

ブザーBIOSについて個別に詳しく解説します.

ブザー	INT 9EH
ブザーON	機能コード00H

エントリ AH =00H

リターン AH =00H(正常終了)

説明 システムが設定した周波数で,ブザーを鳴らします.

システム周波数=1200Hz

ブザー	INT 9EH
ブザーOFF	機能コード01H

エントリ AH =01H

リターン AH =00H(正常終了)

説明 「ブザーON(機能コード00H)」で鳴らしたブザーを止めます.

ブザー	INT 9EH
ブザーON(一定時間)	機能コード02H

エントリ AH =02H

リターン AH =00H(正常終了)

説明 システムが設定した周波数,時間でブザーを鳴らします.

システム周波数=1200Hz
システム時間 =200ms

ブザー	INT 9EH
ブザーON(カウンタ数，指定時間)	**機能コード03H**

エントリ	AH	＝03H
	BX	＝時間
	DX	＝カウンタ数
リターン	AH	＝00H(正常終了)

説　明

指定された周波数，時間でブザーを鳴らします．

時間は，10ms単位で，指定時間が0のときは音は鳴りません．

カウンタ数は，BEEP音の周波数を決める値で，次の計算式によって求められます．

カウンタ数＝基本周波数(19200Hz)÷周波数

ブザー	INT 9EH
ブザー情報の読み取り1	**機能コード04H**

エントリ	AH	＝04H
リターン	AH	＝00H(正常終了)
	AL	＝00H(ブザーが止まっている状態)
		01H(ブザーが鳴っている状態)
	BX	＝時間
	DX	＝カウンタ数

説　明

「ブザーON(カウンタ数，指定時間)(機能コード03H)」で指定した情報(カウンタ数，時間)を読み取ります．

時間は，10ms単位で，カウンタ数はBEEP音の周波数を決める値です．

「ブザーON(カウンタ数，指定時間)(機能コード03H)」で，周波数，時間が指定されていない場合のカウンタ数と時間は，0になっています．

ブザー	INT 9EH
ブザーON(周波数，指定時間)	**機能コード05H**

エントリ　AH　＝05H
　　　　　BX　＝時間
　　　　　DX　＝周波数

リターン　AH　＝00H(正常終了)

説　明

指定された周波数，時間でブザーを鳴らします．

時間は，10ms単位で，指定時間が0のときは音は鳴りません．周波数は，ブザー音の周波数を決める値です．

ブザー	INT 9EH
ブザー情報の読み取り2	**機能コード06H**

エントリ　AH　＝06H

リターン　AH　＝00H(正常終了)
　　　　　AL　＝00H(ブザーが止まっている状態)
　　　　　　　　01H(ブザーが鳴っている状態)
　　　　　BX　＝時間
　　　　　DX　＝周波数

説　明

「ブザーON(周波数，指定時間)(機能コード05H)」で指定した情報(周波数，時間)を読み取ります．

周波数はブザーの周波数を決める値です．時間は10ms単位です．

「ブザーON(周波数，指定時間)(機能コード05H)」で周波数，時間が指定されていない場合の周波数と時間は0になっています．

第14章

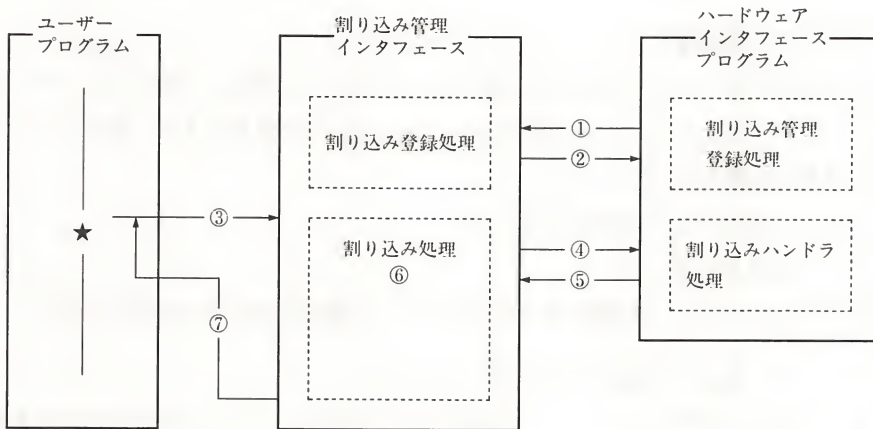
割り込み管理BIOS

割り込み管理 BIOS は，ハードウェア割り込みを管理します．割り込みが発生すると割り込み要因を解析し，その要因に一致した割り込みハンドラに制御を渡します．

この章では，この割り込み管理 BIOS について解説します．

14.1 割り込み管理 BIOS の概要

図II-14-1に割り込みの制御の流れを示します．

▼図II-14-1　割り込み制御の流れ

★は，ハードウェア割り込み発生を示します．

①割り込み管理に，割り込みを登録する．

②指定された割り込みが受け付け可能となる．

③ハードウェアより割り込みが発生すると，割り込み管理 BIOS に制御が渡る．

④割り込み管理は要因を解析し，その要因の登録されている割り込みハンドラに制御を渡す(セグメント間コール)．
⑤割り込みハンドラの処理を行い，終了時に割り込み管理に制御をもどす(セグメント間リターン)．
⑥割り込み管理は，割り込みコントローラに対し，EOI(END Of Interrupt)を返す．
⑦ハードウェア割り込みが発生した時点のプログラムに制御をもどす．

割り込み管理BIOSの使用方法は次のとおりです．

1．割り込み登録処理

次の手順で行います．
この登録処理を行うことにより，指定された要因のハードウェア割り込みが有効となります．

①各ハードウェアの初期化を行う．
②「割り込みデータブロックアドレスの取り出し(機能コード01H)」で，登録済みの割り込みデータブロックアドレスを取り出す．
③「割り込みデータブロックアドレスの登録(機能コード00H)」で，登録する割り込みハンドラのブロックアドレスを設定する．
④「割り込み許可データの取り出し(機能コード03H)」でマスク情報を取り出す．
⑤「割り込み許可データの書き込み(機能コード02H)」で，④で取り出したマスク情報に登録したい要因を追加し，割り込み管理への登録処理を終了する．

2．割り込みハンドラ処理

割り込みが発生すると，その割り込みを発生したデバイスに対応した割り込みハンドラに制御を渡します．このとき，セグメント間呼び出し(far CALL)が使われます．割り込みハンドラでは，次の手順で処理を行ってください．

①ハードウェアの割り込み要因を解除する．
②ハンドラの処理を行う．
③処理終了後は，セグメント間復帰(far RET)をして割り込み管理へ制御をもどす．

割り込みハンドラ処理では次の点に注意してください．

・割り込み管理から割り込みハンドラに制御が渡ったとき，ハンドラ内でのレジスタの保存の必要はありません．保存のためレジスタの退避，復帰を行うと，無駄なオーバーヘッドタイムがかかります．
・同様にオーバーヘッドタイムを少なくするためハンドラ処理はフラグをセットするとか，必要なデータを転送する程度にし，なるべく処理時間を最小限にとどめてください．

・ハードウェア割り込み要因は，必ず解除しなければなりません．解除しない場合，割り込み処理を終了しても，またすぐに同じ割り込みが発生して永久ループとなるため，ユーザープログラムに制御が渡らなくなります．
　解除の方法はデバイスによって異なり，あるレジスタを読み出すとかレジスタの特定ビットを変更するなどの操作が必要です．詳しくは第I部の該当デバイスの説明を参照してください．
・割り込みコントローラに対するEOI　(END Of Interrupt)は行う必要はありません．割り込みハンドラ内では，Iフラグ(インタラプトフラグ)を変えることは，原則としては行ってはいけません．
・各割り込み機能の処理中は，割り込みはすべてマスクされており不可能になっています(ただしNMIはマスク不可能です)．

14.2　割り込み管理BIOS一覧

表II-14-1に割り込み管理BIOSの一覧を示します．

▼表II-14-1　割り込み管理BIOS一覧

機能名称	機能コード
割り込みデータブロックアドレスの登録	00H
割り込みデータブロックアドレスの取り出し	01H
割り込み許可データの書き込み	02H
割り込み許可データの取り出し	03H
割り込みデータブロックテーブルの取り出し	04H

14.3 割り込み管理BIOSリファレンス

割り込み管理BIOSについて個別に詳しく解説します.

割り込み管理	INT AEH
割り込みデータブロックアドレスの登録	機能コード00H

エントリ
AH =00H
DL =割り込み要因コード
DS:DI =割り込みデータブロックアドレス

リターン
AH =00H(正常終了)
02H(割り込み要因コードの誤り)

説　明
割り込みハンドラを指定された要因コード別に登録します.
このオペレーション実行後, 割り込みが発生した場合は, 指定された要因の割り込みデータブロックの割り込みハンドラアドレスを参照し, 割り込みハンドラへ制御を渡します.
割り込み要因コードを示します.

要因コード	IRQ	割り込み要因内容	優先順位
00H	IRQ 0	タイマ	高
01H	IRQ 1	キーボード	↑
02H	IRQ 2	RS-232C	
03H	IRQ 3	拡張RS-232C	
04H	IRQ 4	I/O拡張ユニット	
05H	IRQ 5	I/O拡張ユニット	
06H	IRQ 6	フロッピィディスク制御	
07H	IRQ 7	使用不可(カスケード接続に使用のため)	
08H	IRQ 8	SCSI制御	
09H	IRQ 9	CD-ROM	
0AH	IRQ A	I/O拡張ユニット	
0BH	IRQ B	VSYNC	
0CH	IRQ C	プリンタ制御	
0DH	IRQ D	FM, PCM	
0EH	IRQ E	I/O拡張ユニット	↓
0FH	IRQ F	予約済	低

割り込みデータブロックの内容を示します．

(DS:DI)

0	E	B	0
1	E	B	0
2	E	D	割り込みハンドラアドレス

割り込みハンドラアドレスは，割り込みが発生したときに制御を渡す割り込みハンドラのアドレスを指定します．

割り込みハンドラを呼び出す際には，セグメント間呼び出し(far CALL)が使用されるので，リターン時にはセグメント間復帰(far RET)を使用して復帰させます．このとき，レジスタを復旧する必要はありません．

割り込み管理　INT AEH

割り込みデータブロックアドレスの取り出し　機能コード01H

エントリ

AH　＝01H

DL　＝割り込み要因コード

リターン

AH　＝00H(正常終了)

　　02H(割り込み要因コードの誤り)

DS:DI　＝割り込みデータブロックアドレス

説　明

割り込みデータブロックテーブル中で，指定された要因コードが現在登録されているデータブロックのアドレスを取り出します．

割り込みハンドラの登録を行う際，登録以前に別なハンドラが登録されていて，その機能を使用したい場合，この機能でデータブロックアドレスを取り出しておく必要があります．

割り込みハンドラで，登録以前に登録されているハンドラに制御を渡したいときは，この機能でデータブロックアドレスを取り出しておき，ハンドラの処理を終了した後，取り出しておいたデータブロック内の割り込みハンドラアドレスにジャンプすることにより，すでに登録していた割り込みハンドラに制御を渡すことが可能となります．

使用した割り込みハンドラの登録を解除するときには，新しいデータブロックのアドレスを「割り込みデータブロックアドレスの登録(機能コード00H)」で書き換えてください．

割り込み管理	INT AEH
割り込み許可データの書き込み	機能コード02H

エントリ　AH　＝02H

DS : DI　＝割り込み許可データパラメータブロックアドレス

リターン　AH　＝00H（正常終了）

説　明　割り込み許可データの値を書き込みます．

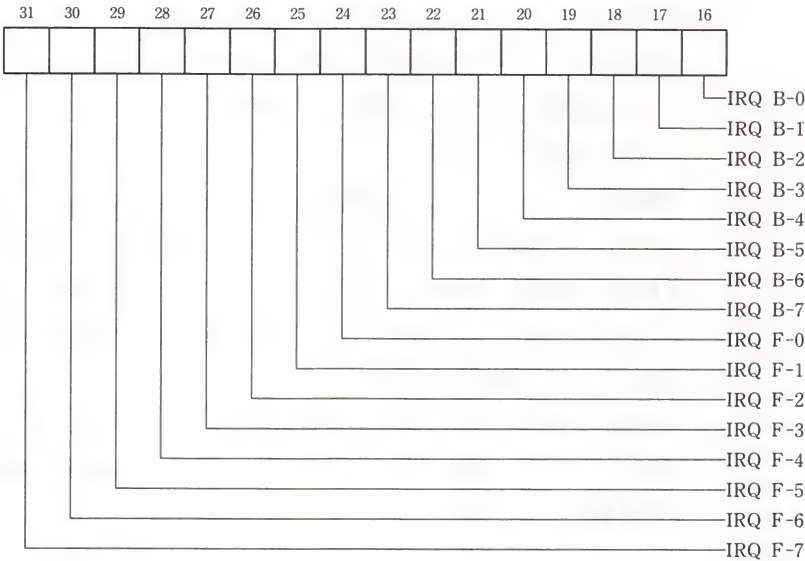
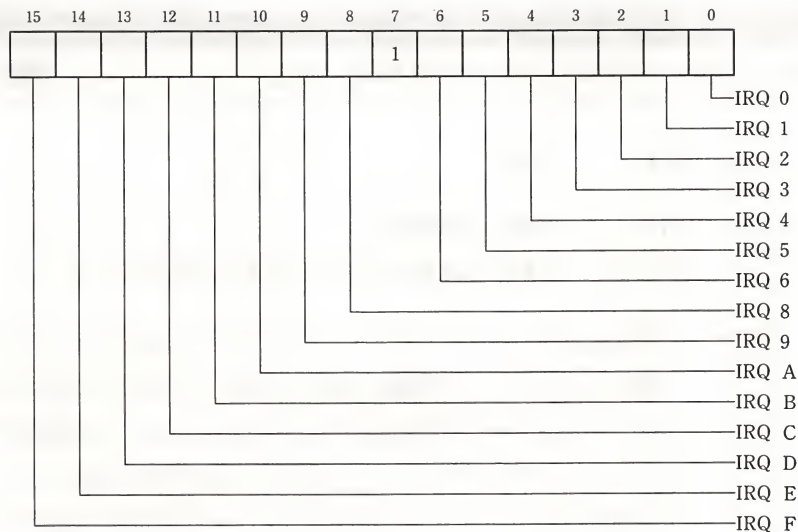
割り込み許可データの書き込みを行う場合,「割り込み許可データの取り出し（機能コード03H）」で現在の状態を参照し，必要なビット（各ビットは0が割り込み不許可，1が割り込み許可）を操作してから書き込みを行ってください．

この機能を使用するときは，事前に「割り込みデータブロックアドレスの登録（機能コード00H）」で，各ハードウェアの初期化が行われていなければなりません．

割り込み許可データパラメータの形式を示します．

(DS:DI)

E	B	ビット　31～24
E	B	ビット　23～16
E	B	ビット　15～ 8
E	B	ビット　7～0

割り込み管理の初期化が終了したときは，すべての割り込みが不許可になっています．

割り込み管理　INT AEH

割り込み許可データの取り出し　機能コード03H

エントリ　AH　＝03H

DS：DI　＝割り込み許可データパラメータブロックアドレス

リターン　AH　＝00H（正常終了）

説　明

割り込み許可データの値を取り出します．

割り込み許可データの書き込みを行う場合，この機能で割り込み許可データを取り出し，ビット操作(各ビットは0が割り込み不許可，1が割り込み許可)後，書き込みを行います．

割り込み許可データパラメータの形式を示します．

(DS:DI)

R	B	ビット　31〜24
R	B	ビット　23〜16
R	B	ビット　15〜 8
R	B	ビット　7〜0

割り込み管理の初期化が終了したときは，すべての割り込みが不許可になっています．

割り込み管理	INT AEH
割り込みデータブロックテーブルの取り出し	機能コード04H

エントリ　AH　＝04H

リターン　AH　＝00H（正常終了）
DS : DI　＝割り込みデータブロックテーブルアドレス

説　明　割り込みデータブロックテーブルのアドレスを取り出します．

割り込みハンドラの登録が複数ある場合，要因別に登録すると複数回，ハンドラアドレスの登録を呼ばなければいけませんが，この機能を利用してデータブロックテーブルのアドレスを参照し，相対位置に直接データブロックアドレスを書き込むことにより，複数個のハンドラを同時に登録することができます．

割り込みデータブロックテーブルの形式を示します．

(DS : DI)

オフセット	サイズ	内容
0	D	要因00H　オフセットアドレス
		要因00H　セグメントアドレス
	D	：
18	D	要因06H　オフセットアドレス
		要因06H　セグメントアドレス
1C	D	0
		0
20	D	要因08H　オフセットアドレス
		要因08H　セグメントアドレス
	D	：
40	D	要因10H　オフセットアドレス
		要因10H　セグメントアドレス
	D	：
7C	D	要因1FH　オフセットアドレス
		要因1FH　セグメントアドレス

第15章

サービスルーチンと拡張サービスルーチン

よく使用される機能をまとめたサービスルーチンと，TOWNS OSのシステム情報を取得，あるいは設定する機能などを記述した拡張サービスルーチンが用意されています．この章では，これらについて解説します．

15.1 サービスルーチン，拡張サービスルーチン一覧

表II-15-1にサービスルーチン，拡張サービスルーチンの一覧を示します．

▼表II-15-1　サービスルーチン，拡張サービスルーチン一覧

サービスルーチン

機能名称	機能コード
JISからシフトJISへの変換	00H
シフトJISからJISへの変換	01H
CPUタイプの読み取り	02H
JISからシフトJISへの変換2	03H
シフトJISからJISへの変換2	04H
機器情報の読み取り	05H

拡張サービスルーチン

機能名称	機能コード
システム情報の取得	00H
カットシートフィーダ制御の設定	01H

15.2 サービスルーチン，拡張サービスルーチンリファレンス

サービスルーチンと拡張サービスルーチンについて，個別に詳しく解説します．

なお，各機能が正常に終了したか否かは，AHレジスタに通知されます．次のエラーコードについては，各機能共通です．

AH　=00H（正常終了）
　　 01H（未定義ファンクションコードのエラー）

サービスルーチン　INT AFH
JISからシフトJISへの変換　機能コード00H

エントリ　AH　=00H
　　　　　DX　=JIS漢字コード

リターン　AH　=00H（正常終了）
　　　　　DX　=シフトJIS漢字コード

説　明　JIS漢字コードを，シフトJIS漢字コードに変換します．
DXに設定する値の形式を示します．

DH	DL
漢字上位バイト	漢字下位バイト

サービスルーチン　INT AFH
シフトJISからJISへの変換　機能コード01H

エントリ　AH　=01H
　　　　　DX　=シフトJIS漢字コード

リターン　AH　=00H（正常終了）
　　　　　　　02H（シフトJIS漢字コードの誤り）
　　　　　DX　=JIS漢字コード

説　明　　シフト JIS 漢字コードを，JIS 漢字コードに変換します．
DX に返される値の形式を示します．

DH	DL
漢字上位バイト	漢字下位バイト

サービスルーチン　INT AFH

CPU のタイプの読み取り　機能コード02H

エントリ　AH　＝00H

リターン　AH　＝00H（正常終了）
DH　＝00H
DL　＝01H（8086）
02H（80186）
03H（80286）
04H（80286-0WAIT）
05H（80386）
06H（80486）
07H（80386SX）

説　明　　CPU のタイプを読み取ります．

サービスルーチン　INT AFH

JIS からシフト JIS への変換 2　機能コード03H

エントリ　AH　＝03H
DX　＝JIS 漢字コード

リターン　AH　＝00H（正常終了）
DX　＝シフト JIS 漢字コード

説　明　　JIS 漢字コードを，シフト JIS 漢字コードに変換します．
DX に設定する値の形式を示します．

DH	DL
漢字下位バイト	漢字上位バイト

サービスルーチン	INT AFH
シフトJISからJISへの変換2	機能コード04H

エントリ	AH	＝04H
	DX	＝シフトJISコード
リターン	AH	＝00H（正常終了）
		02H（シフトJISコードの誤り）
	DX	＝JIS漢字コード

説　明　シフトJIS漢字コードを，JIS漢字コードに変換します．

DXに返される値の形式を示します．

DH	DL
漢字下位バイト	漢字上位バイト

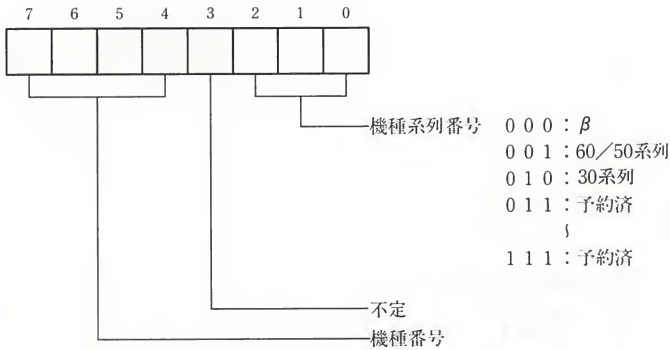
サービスルーチン	INT AFH
機器情報の読み取り	機能コード05H

エントリ	AH	＝05H
	DS : DI	＝機器情報エリアアドレス
リターン		
	AH	＝00H（正常終了）

説　明　機器情報を指定されたエリアに設定します．機器情報エリアとして，16バイトが必要です．

機器情報エリアの形式を示します．

(DS:DI)			
0	R	B	機種 ID
1	R	B	CPU タイプ
2	R	B	ディスプレイタイプ
3	R	B	ディスプレイ解像度
4	R	B	キーボードタイプ
5	R	W	BIOS バージョン
7	R	B	数値演算プロセッサの有無
8	R	B	DOS レベル番号
9	R	B	DOS レベル番号サフィックス
10	R	B	内蔵フロッピィ識別情報
11	R	B	拡張ディスプレイ機能
12	R	B	CPU クロックレート
13	R	B	メモリウェイト数
14	R	B	アスペクト比
15	R	B	予約済

●機種 ID

機種系列番号・機種番号と機種名との関係を以下に示します．

		機種系列番号		
		0 0 0	0 0 1	0 1 0
機種番号	0 0 0 0	FM16β	FMR-60/50＊	FMR-30
	0 0 1 0	予約済	FMR-70＊	FMR-30HX
	0 0 1 1	予約済	FMR-50S	FMR-30BX
	0 1 0 0	予約済	FMR-50LT	予約済
	0 1 0 1	予約済	FM TOWNS	予約済
	∫		予約済	
	1 1 1 1	予約済	予約済	FMR-10LT

＊FMR-60/50 ………FMR-60HX/60FX/60/50HX/50FX/50/50Λ/50LX/50TX/50FV/50HV/50ΛLX/CARD/50NB1/50SHX/50SFX/50SIIFX/50SIIHX
＊FMR-70……………FMR-70HX3/70HX2/70HX1/70/70Σ/70HX2S

●CPU タイプ

01H＝8086

02H＝80186

03H＝80286

05H＝80386

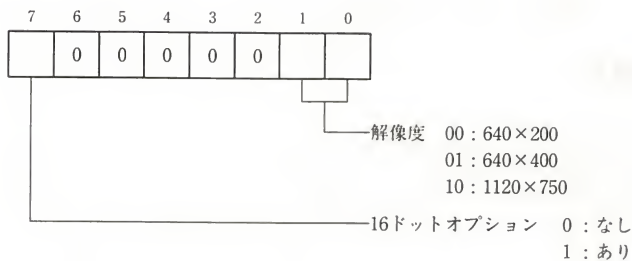
06H＝80486

07H＝80386SX

●ディスプレイタイプ

00H＝カラー

01H＝白黒

●解像度

●キーボードタイプ

00H＝JIS キーボード

01H＝親指キーボード

●BIOS のバージョン

2バイトのバイナリ表現です.

●数値プロセッサの有無

00H＝数値プロセッサなし

01H＝数値プロセッサあり

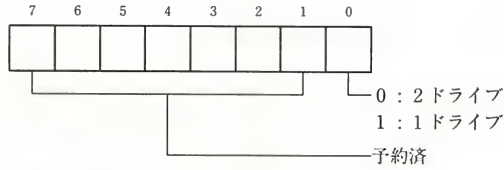
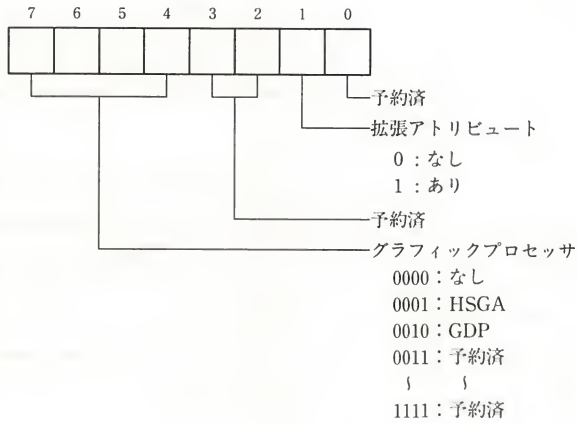
●DOS レベル番号

MS-DOS システムのレベル番号をバイナリで表します.

●DOS レベル番号サフィックス

MS-DOS システムのレベル番号サフィックスをバイナリで表します.

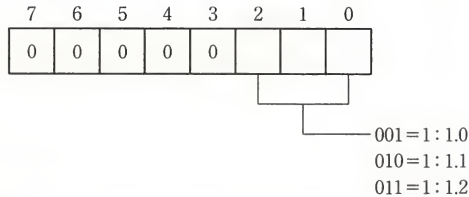
●内蔵フロッピィ識別情報

本体に内蔵されているフロッピィディスクドライブについての情報を示します．

●拡張ディスプレイ機能

●CPU クロックレート

MHz 単位のクロックレートをバイナリで表します．

●メモリウェイト

メモリのウェイト数をバイナリで表します．

●アスペクト比

拡張サービスルーチン	INT 8EH
システム情報の取得	機能コード00H

エントリ　AH　＝00H

DS：DI　＝システム情報取得エリアアドレス

リターン　AH　＝00H（正常終了）

説　明　指定されたシステム情報取得エリアにシステム情報を書き込みます.
システム情報取得エリアは200バイト必要です.
システム情報取得エリアの形式を示します.

●BOOT デバイスタイプ

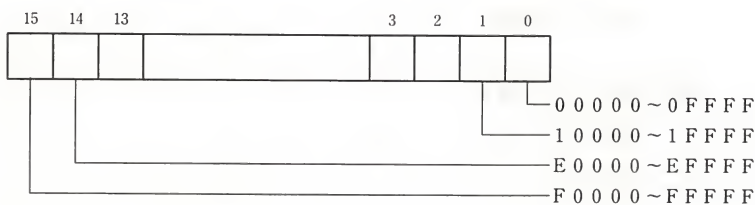
01H＝ハードディスク

02H＝1MB フロッピィディスク

●BOOT ユニット番号

1MB フロッピィディスク：0〜3

ハードディスク：0〜6

●メモリ実装フラグ

各ビットが64KB 単位にメモリの実装を示しています．ビット ON でメモリが存在していることを示します．

●シングルドライブ

00H≠　シングルオペレーションあり．

●CP-MGR インストールフラグ

ビット 0＝ON　インストールされている．

●VJE-α インストールフラグ

ビット 0＝ON　インストールされている．

●RAM DISK サイズ

指定されている RAM DISK のサイズを KB で示す．

●かな漢字変換インストールフラグ

ビット 0＝ON　インストールされている．

●未割当領域先頭アドレス

拡張アドレス(100000H 以降)で，MS-DOS システムに割り当てられていない領域の先頭アドレスを示す(32 ビット絶対アドレス表現)．なお，0 のとき未割当領域なし．

●未割当領域サイズ

拡張アドレス(100000H 以降)で，MS-DOS システムに割り当てられていない領域のサイズを示す(KB単位)．なお，0のとき未割当領域なし．

●ドライブ種別

00H＝1MB フロッピィ

01H＝予約済

02H＝ハードディスク

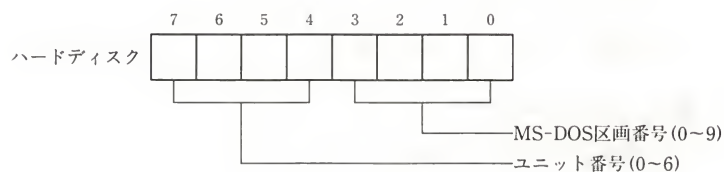
03H＝RAM ディスク

FFH＝未登録

●物理ドライブ番号

1MB フロッピィディスク：0～2

RAM ディスク：0

ハードディスク

●プリンタ種別

01H＝漢字プリンタ

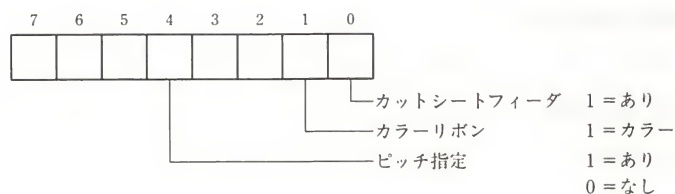
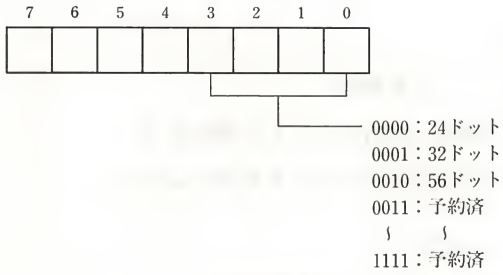
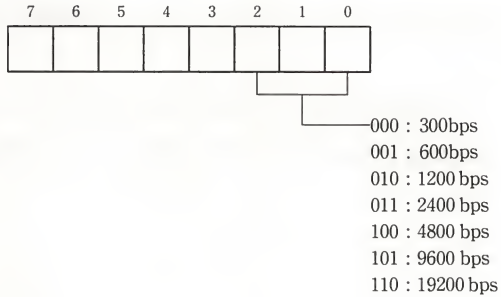
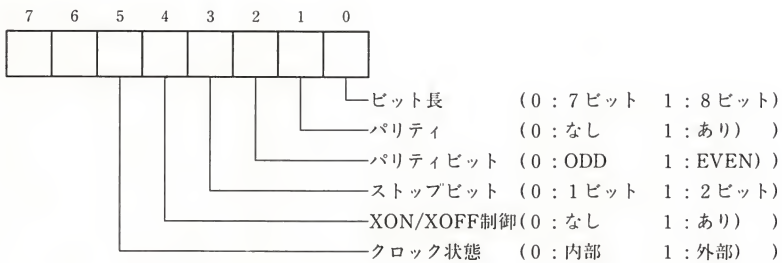
●プリントモード

(漢字プリンタ)

00H＝非漢字モード

01H＝漢字モード，横書き，縮小 ANK 文字

03H＝漢字モード，横書き，標準 ANK 文字

●プリンタオプション

●プリンタタイプ

●RS-232C ボーレート

●通信モード

●送受信タイムアウト値

0　：即時復帰

65535　：完了復帰

1～65534　：指定時間(単位10ms)内に完了しない場合，強制復帰

拡張サービスルーチン	INT 8EH
カットシートフィーダ制御の設定	**機能コード01H**

エントリ

AH　＝01H
AL　＝00H（プリンタ番号）
DL　＝00H（吸入／排出コマンドを発行する）
　　　01H（吸入／排出コマンドを発行しない）

リターン

AH　＝00H（正常終了）
　　　02H（無効なプリンタ番号指定）
　　　03H（無効な吸入／排出コマンド指定）

説　明

TOWNS MENUで設定された，カットシートフィーダ制御を有効にするか無効にするかを設定します．

このファンクションはMS-DOSのファンクションを使用している場合に有効です．また，一度制御を有効にするとプリンタをリセットするまで制御を無効にできません．

第16章

システム情報BIOS

システム情報BIOSは，一部のネイティブBIOSで与えられた設定値を読み出すためのもので，グラフィック，マウス，サウンドの各BIOSに対応する機能がサポートされています.

読み出される内容は，これらのBIOSで指示された値がシステム領域に転送され，現在値が残っているものが対象となります．したがって，ユーザーがBIOSを使用せずに設定した場合は，読み出し結果は意味がないので注意が必要です.

16.1 システム情報BIOS一覧

システム情報BIOSは，次のグループに分類することができます.

1. グラフィックBIOS設定値読み取り

現在の画面モード，書き込みページ，表示ページ，表示開始位置，パレットを読み取ることができます.

2. マウスBIOS設定値読み取り

マウスカーソルの表示／消去状態，水平移動範囲，垂直移動範囲，ユーザー定義サブルーチンの設定，マウスカーソルの移動量，画面モード，書き込みページ，加速度検出状態，動作状態を読み取ることができます.

3. サウンドBIOS設定値読み取り

電子ボリュームの設定状態，電子ボリュームミュート設定状態を読み取ることができます.

4. 解像度ハンドルの取得

解像度ハンドルとは，FMTOWNSの高解像度化に伴い，画面の解像度などの情報を取得し細かな画面制御に反映できるようにするために，システムから与えられる仮定値をいいます.

この値は，本章で述べるオペレーションでいろいろな角度から取得でき，第2章のグラフィッ

ク BIOS の解像度ハンドルによる「仮想画面の設定（機能コード 1CH）」などで利用します．その際，取得した解像度ハンドルをもとに，解像度を設定することができます．

また，画面表示の高速化のため VRAM への直接アクセスを行う際に必要な情報として，VRAM 先頭アドレスなどの値が得られるため，これを利用すれば，機種や画面モードに依存しないプログラムを作成することができます．

表 II-16-1 に，システム情報 BIOS 一覧を示します．

▼表 II-16-1　システム情報 BIOS 一覧

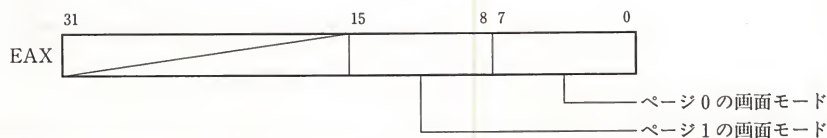
機能名称	機能コード
仮想画面の読み取り	01H
書き込みページの読み取り	02H
表示ページの読み取り	03H
表示開始位置の読み取り	04H
パレットレジスタの読み取り	05H
画面モードに関する情報の取得	0AH
現在の表示画面サイズの取得	0BH
表示/消去状態の読み取り	11H
水平移動範囲の読み取り	12H
垂直移動範囲の読み取り	13H
サブルーチンの読み取り	14H
パルス数/画素比の読み取り	15H
仮想画面の読み取り	16H
書き込みページの読み取り	17H
ボタン左右入れ換え状態の読み取り	18H
加速度検出状態の読み取り	19H
動作状態の読み取り	1AH
電子ボリュームの設定状態の読み取り	21H
電子ボリュームミュート設定状態の読み取り	22H
現在登録されている全サウンド ID の取得	23H
音声モード使用チャネル数の取得	24H
割り込み管理システム情報の設定	30H
割り込み管理システム情報の取得	31H
パラメータによる解像度ハンドルの取得	40H
ページ指定による解像度の取得	41H
ピクセル数（色数）による解像度の取得	42H
画面モード番号による解像度ハンドルの取得	43H
表示設定可能ページの取得	44H
パレット有効ビットの取得	45H
VRAM 有効ビットの取得	46H

16.2　システム情報BIOSリファレンス

システム情報　1C0H
仮想画面の読み取り　機能コード01H

エントリ	AH	＝01H
リターン	EAX	＝画面モード

説　明

ページ0およびページ1の現在の画面モードを読み取ります．結果は図のような形式でEAXに収容されます．

システム情報　1C0H
書き込みページの読み取り　機能コード02H

エントリ	AH	＝02H
	CX	＝作業領域のセグメント
	EDX	＝作業領域のアドレス
リターン	EAX	＝書き込みページ

説　明

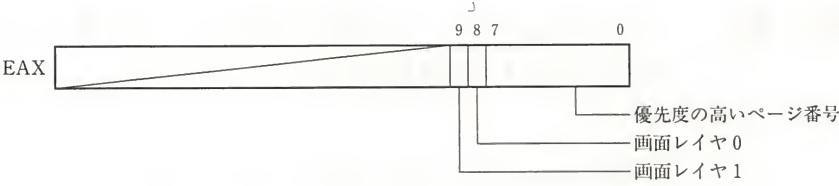
指定されたEGB作業域内の現書き込みページを読み取ります．

CX＝0にした場合は，EGBが実際にハードウェア上で設定した書き込みページを読み取ります．

システム情報	1C0H
表示ページの読み取り	機能コード03H

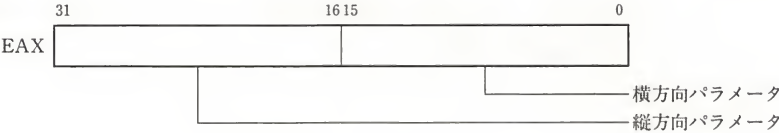
エントリ　AH　=03H

リターン　EAX　=表示ページ，プライオリティ

説　明　現在の表示ページとその優先順位を読み取ります．結果は図のような形式でEAXに収容されます．

システム情報	1C0H
表示開始位置の読み取り	機能コード04H

エントリ　AH　=04H
CX　=ページ番号(0，1)
CH　=モード(0〜3)

リターン　EAX　=表示開始位置など

説　明　現在の表示開始位置などを読み取ります．結果は図のような形式でEAXに収容されます．モードの意味については，グラフィックBIOSの「表示開始位置の設定(機能コード02H)」を参照してください．

システム情報	1C0H
パレットレジスタの読み取り	機能コード05H

エントリ　AH　＝05H
AL　＝ページ番号(0, 1)
DS：EDX ＝読み取り先アドレス

リターン　EAX　＝0(正常終了時)
＝−1(パレットの存在しない画面モードでこのコマンドを実行したとき．たとえば，32768色の画面モードのときなど)

説　明　現在のパレットの設定状態を取得します．取得データは，指定された読み取り先アドレスに以下のデータ形式で出力されます．

赤，青，緑色の階調データは，以下のような意味を持ちます．

システム情報	1C0H
画面モードに関する情報の取得	機能コード0AH

エントリ
AH　　　=0AH
AL　　　=画面モード番号（1～18）

リターン
EAX　　=0（正常終了）
EDX　　=仮想画面サイズ（X，Y）
EBX　　=表示画面サイズ（X，Y）
ECX　　=同時発色数（16，256，32768）

説　明
指定された画面モードの仮想画面サイズ，表示画面サイズ，同時発色数をレジスタに取得します．

画面サイズを取得した結果は，次のような形式になっています．

	32　　　　16	15　　　　0
EDX	仮想画面の大きさ（Y）	仮想画面の大きさ（X）
EBX	表示画面の大きさ（Y）	表示画面の大きさ（X）

システム情報	1C0H
現在の表示画面サイズの取得	機能コード0BH

エントリ
AH　　　=0BH
AL　　　=ページ番号（0，1）
DS：ECX =取得バッファのアドレス

リターン
EAX　　=0（正常終了）
　　　　−1（エラー）

説　明
表示中の画面のピクセル数を取得するオペレーションです．このとき，画面倍率は関係しません．

画面サイズを取得した結果は，次のような形式になっています．

(DS：ECX)

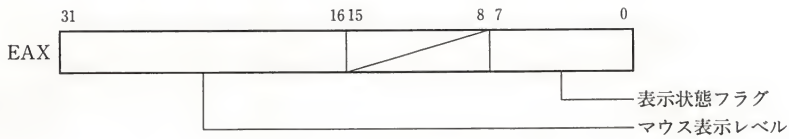
0	DW	表示開始位置X
4	DW	表示開始位置Y
8	DW	画面サイズX
12	DW	画面サイズY

システム情報　1C0H

表示／消去状態の読み取り　機能コード11H

エントリ　AH　＝11H

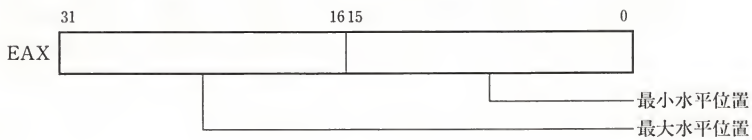
リターン　EAX　＝表示フラグ，マウス表示レベル

説　明　現在のマウス表示状態とマウス表示レベルを読み取ります．結果は，図のような形式でEAXに収容されます．

システム情報　1C0H

水平移動範囲の読み取り　機能コード12H

エントリ　AH　＝12H

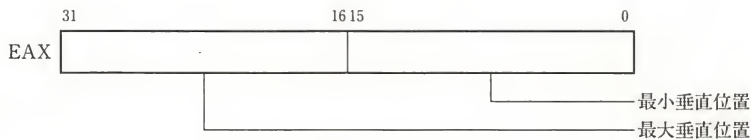
リターン　EAX　＝最小水平位置，最大水平位置

説　明　現在のマウスの水平方向の移動範囲を読み取ります．結果は，図のような形式でEAXに収容されます．

システム情報	1C0H
垂直移動範囲の読み取り	**機能コード13H**

エントリ　AH　＝13H

リターン　EAX　＝最小垂直位置，最大垂直位置

説　明

現在のマウスの垂直方向の移動範囲を読み取ります．結果は，図のような形式で EAX に収容されます．

システム情報	1C0H
サブルーチンの読み取り	**機能コード14H**

エントリ　AH　＝14H

リターン　EAX　＝呼び出し条件(定義時に指定した値)
　　　　　ECX　＝サブルーチンのセグメント
　　　　　EDX　＝サブルーチンのアドレス

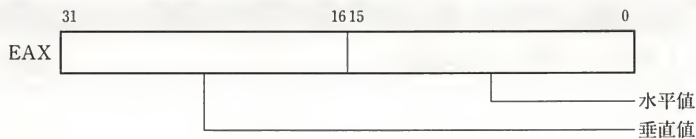
説　明

マウスのユーザー定義サブルーチンの呼び出し条件とセグメント，アドレスを読み取り，結果をそれぞれ EAX，ECX，EDX に収容します．

システム情報	1C0H
パルス数／画素比の読み取り	機能コード15H

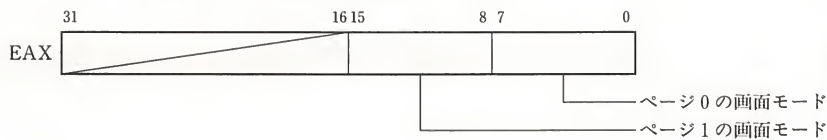
エントリ　AH　＝15H

リターン　EAX　＝1ドットの移動に必要なパルス数

説　明　マウスカーソルを1ドット移動するのに必要なパルス数が読み取られます．結果は，図のような形式でEAXに収容されます．

システム情報	1C0H
仮想画面の読み取り	機能コード16H

エントリ　AH　＝16H

リターン　EAX　＝画面モード番号

説　明　現在のマウスの画面モードを読み取ります．結果は，図のような形式でEAXに収容されます．

システム情報	1C0H
書き込みページの読み取り	**機能コード17H**

エントリ　AH　＝17H

リターン　EAX　＝書き込みページ

説　明　現在のマウスの書き込みページを読み取り，結果を EAX に収容します．

システム情報	1C0H
ボタン左右入れ換え状態の読み取り	**機能コード18H**

エントリ　AH　＝18H

リターン　EAX　＝入れ換え状態(0：通常，1：入れ換え)

説　明　現在のマウスボタンの左右入れ換え状態を読み取り，結果を EAX に収容します．

システム情報	1C0H
加速度検出状態の読み取り	**機能コード19H**

エントリ　AH　＝19H

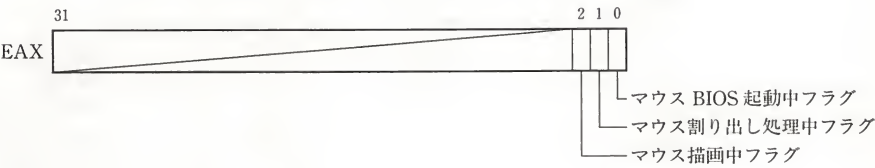
リターン　EAX　＝加速度検出状態(0：無効，1：有効)

説　明　現在のマウスの加速度検出状態を読み取り，結果を EAX に収容します．

システム情報	1C0H
動作状態の読み取り	機能コード1AH

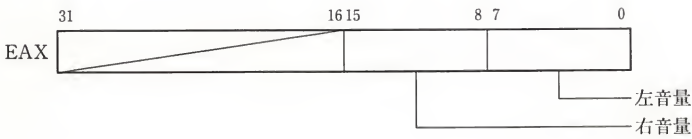
エントリ　AH　＝1AH

リターン　EAX　＝マウスの動作状態(各ビットとも0：非動作，1：動作中)

説　明　現在のマウスの動作状態を読み取ります．結果は，図のような形式でEAXに収容されます．

システム情報	1C0H
電子ボリュームの設定状態の読み取り	機能コード21H

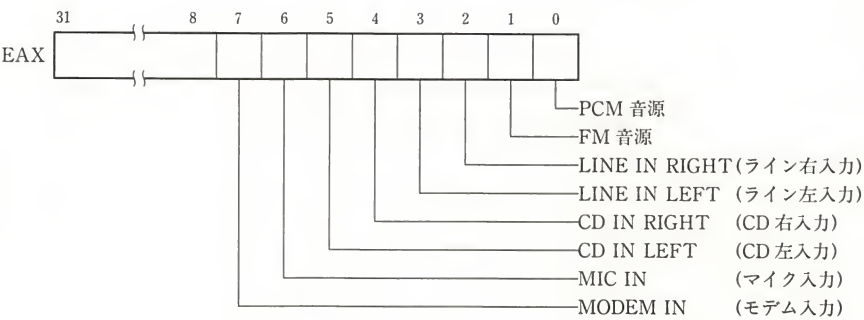
エントリ　AH　＝21H
　　　　　AL　＝ボリューム番号(0～3)

リターン　EAX　＝電子ボリュームの設定値(0～127)

説　明　指定された電子ボリュームの設定値を読み取ります．結果は，図のような形式でEAXに収容されます．

システム情報	1C0H
電子ボリュームミュート設定状態の読み取り	機能コード22H

エントリ　AH　＝22H

リターン　EAX　＝ミュート設定状態(各ビットとも0：ミュート，1：解除)

説　明　電子ボリュームのミュート／解除の設定値を読み取ります．

システム情報	1C0H
現在登録されている全サウンドIDの取得	**機能コード23H**

エントリ　AH　＝23H
DS：EDX ＝サウンドID取得バッファのアドレス

リターン　EAX　＝0（正常終了）

説　明

現在登録されている全サウンドIDについて，個別のID値とそれに対応するデータ長を読み出し，バッファ（メモリ）に展開します.

バッファの大きさは，ひとつのIDについて8バイト×最大登録数（128個分）だけ必要なので，1024バイト用意します.

メモリに展開した結果は，次のような形式になっています．なお，サウンドがまったく登録されていないときは，サウンド1のID値に0が入ります.

(DS：EDX)

オフセット		
0	D	サウンド1のID
4	D	サウンド1のデータ長
8	D	サウンド2のID
12	D	サウンド2のデータ長
16		
	⋮	
1016	D	サウンド128のID
1020	D	サウンド128のデータ長

システム情報	1C0H
音声モード使用チャネル数の取得	**機能コード24H**

エントリ　AH　=24H

リターン　EAX　=音声モードのチャネル数

説　明　現在音声モードに設定されているチャネル数を，EAXに取得します．

システム情報	1C0H
割り込み管理システム情報の設定	**機能コード30H**

エントリ　AH　=30H

AL　=設定番号

1：ネイティブ割り込みベクタアドレス

2：ネイティブ割り込みベクタセレクタ

3：リアル割り込みベクタアドレス

4：リアル割り込みベクタセレクタ

5：マウスのカウント値設定

注）「ネイティブ」とは，プロテクトモードを指します．

EDX　=設定する値

リターン　EAX　=0（正常終了）

説　明　設定番号で指定された割り込み管理の情報項目について，設定値がシステムに書き込まれます．即ち，以後システムがこれらの値を参照すると，その値が使われます．

システム情報	1C0H
割り込み管理システム情報の取得	機能コード 31H

エントリ　AH　＝31H

AL　＝設定番号

1：ネイティブ割り込みベクタアドレス

2：ネイティブ割り込みベクタセレクタ

3：リアル割り込みベクタアドレス

4：リアル割り込みベクタセレクタ

5：マウスの設定カウント値

注）「ネイティブ」とは，プロテクトモードを指します．

リターン　EAX　＝0（正常終了時）

EDX　＝取得した値

説　明　設定番号で指定された割り込み管理の情報項目について，現在システムに設定されている値が EDX に読み出されます．

システム情報	1C0H
パラメータによる解像度ハンドルの取得	機能コード 40H

エントリ
AH　　　＝40H
AL　　　＝ビデオモード（0：無効，1：有効）
CL　　　＝設定する画面総数（1，2）
DS：EDX ＝パラメータのアドレス

リターン
EAX　　＝－1 以外（取得した解像度ハンドル（正常終了））
　　　　　－1（エラー）

説　明
　1画面当たり 32 バイトで，画面総数だけのパラメータを並べ，それに対応する解像度ハンドルを取得します．パラメータは，勝手に設定すると今後の解像度の拡大に対応できなくなるため，ほかのオペレーションにより BIOS 経由で取得した値を使用することが望まれます．
　また，解像度ハンドルの値は，あくまで仮定値であって OS のレベルや実行状態で変動するため，この値を固定的に運用することはできません．

パラメータ形式

ページ 0 のパラメータ
(DS：EDX)

オフセット	型	内容
0	DW	仮想画面 X 方向解像度（ピクセル）
4	DW	仮想画面 Y 方向解像度（ピクセル）
8	DW	表示画面 X 方向解像度（ピクセル）
12	DW	表示画面 Y 方向解像度（ピクセル）
16	DW	VRAM ピクセルのビット数
20	DW	ライン当たりバイト数
24	DW	VRAM 先頭オフセット
28	W	VRAM セレクタ
30	W	予約済

ページ 1 のパラメータ（画面総数＝2 のとき追加）

オフセット	型	内容
32	DW	仮想画面 X 方向解像度（ピクセル）

⋮

（ページ 0 と同様）

システム情報	1C0H
ページ指定による解像度の取得	機能コード 41H

エントリ　AH　　　=41H
　　　　　AL　　　=ページ番号（1，2）
　　　　　DS：ECX =取得バッファのアドレス

リターン　EAX　　　=－1以外（取得した解像度ハンドル（正常終了））
　　　　　　　　　　－1（エラー）

説　明　現在設定されている画面の解像度情報を32バイトで取得します．内容は図のとおりです．

同時に取得される解像度ハンドルの値は，あくまで仮定値であってOSのレベルや実行状態で変動するため，この値を固定的に運用することはできません．

取得バッファのデータ形式

(DS：ECX)

0	DW	仮想画面X方向解像度（ピクセル）
4	DW	仮想画面Y方向解像度（ピクセル）
8	DW	表示画面X方向解像度（ピクセル）
12	DW	表示画面Y方向解像度（ピクセル）
16	DW	VRAMピクセルのビット数
20	DW	ライン当たりバイト数
24	DW	VRAM先頭オフセット
28	W	VRAMセレクタ
30	W	予約済

システム情報 1C0H

ピクセル数（色数）による解像度の取得 機能コード 42H

エントリ
AH =42H
AL =ビデオモード（0：無効，1：有効）
DS：ECX =取得バッファのアドレス
DS：EDX =パラメータのアドレス

リターン
EAX =－1 以外（取得した解像度ハンドル（正常終了））
－1（エラー）

説 明
パラメータで指定されたピクセル数対応の最大解像度を取得するオペレーションです．取得される情報はページあたり 32 バイトで，総ページ数が 2 のときは 2 ページ分の領域が必要となります．

また，解像度ハンドルの値は，あくまで仮定値であって OS のレベルや実行状態で変動するため，この値を固定的に運用することはできません．

パラメータ形式

(DS：EDX)

0	DW	総ページ数（1 または 2）
4	DW	ページ 0 のピクセル数
8	DW	ページ 1 のピクセル数

取得バッファのデータ形式

ページ 0 の情報

(DS：ECX)

0	DW	仮想画面 X 方向解像度（ピクセル）
4	DW	仮想画面 Y 方向解像度（ピクセル）
8	DW	表示画面 X 方向解像度（ピクセル）
12	DW	表示画面 Y 方向解像度（ピクセル）
16	DW	VRAM ピクセルのビット数
20	DW	ライン当たりバイト数
24	DW	VRAM 先頭オフセット
28	W	VRAM セレクタ
30	W	予約済

ページ 1 の情報（画面総数=2 のとき追加）

32	DW	仮想画面 X 方向解像度（ピクセル）
		⋮

（ページ 0 と同様）

システム情報　1C0H
画面モード番号による解像度ハンドルの取得　機能コード 43H

エントリ
AH　=43H
DS：ECX =取得バッファのアドレス
DS：EDX =パラメータのアドレス

リターン
EAX　=−1以外（取得した解像度ハンドル（正常終了時））
　−1（エラー）

説　明
パラメータで指定された画面モード番号対応の解像度ハンドルを取得するオペレーションです．同時に，取得バッファに図示の形式の内容が転送されます．取得される情報はページあたり 32 バイトで，総ページ数が 2 のときは 2 ページ分の領域が必要となります．

また，解像度ハンドルの値は，あくまで仮定値であって OS のレベルや実行状態で変動するため，この値を固定的に運用することはできません．

パラメータ形式

(DS：EDX)

0	DW	総ページ数（1 または 2）
4	DW	ページ 0 の画面モード番号
8	DW	ページ 1 の画面モード番号

取得バッファのデータ形式

ページ 0 の情報

(DS：ECX)

0	DW	仮想画面 X 方向解像度（ピクセル）
4	DW	仮想画面 Y 方向解像度（ピクセル）
8	DW	表示画面 X 方向解像度（ピクセル）
12	DW	表示画面 Y 方向解像度（ピクセル）
16	DW	VRAM ピクセルのビット数
20	DW	ライン当たりバイト数
24	DW	VRAM 先頭オフセット
28	W	VRAM セレクタ
30	W	予約済

ページ1の情報（画面総数=2のとき追加）

（ページ0と同様）

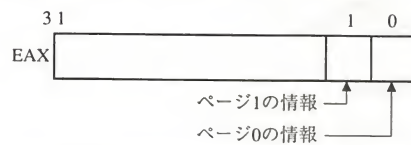
システム情報 1C0H

表示設定可能ページの取得 機能コード44H

エントリ	AH	=44H
	DX	=解像度ハンドル
リターン	EAX	=－1以外（表示可能ページ（正常終了））
		－1（エラー）

説　明　ほかのオペレーションで取得した解像度ハンドルを指定して，現在設定されている画面の各ページが表示可能かどうかを参照します．現在表示されている画面のものを参照するときは，解像度ハンドルの指定を0とします．

参照された内容は，次のビット構成で示されます．

表示可能ページの情報

（各ビットの内容=0：表示不可能，1：表示可能）

システム情報	1C0H
パレット有効ビットの取得	機能コード 45H

エントリ　AH　=45H
DX　=解像度ハンドル
DS：ECX　=取得バッファのアドレス

リターン　EAX　=0（正常終了）
−1（エラー）

説　明

ほかのオペレーションで参照した解像度ハンドルを使って，現在設定されている画面のパレット有効ビットを取得するオペレーションです．現在表示されている画面のものを参照するときは，解像度ハンドルの指定を0とします．

取得バッファは，1ページ当たり8バイト必要です．もし，総ページ数がプログラミング時点で確定しない場合で，静的に割り付けするときは最大2ページ分用意するのが適当です．ただし，ページ0が表示不可能でページ1が表示可能な場合，バッファの先頭からページ1の内容が入ります．

「表示設定可能ページの取得（機能コード44H）」で，総ページ数を表示可能ページの合計により算出した結果によって動的に割り付けする場合は，そのページ数分のバイト数を確保します．

取得した内容のうち，パレット／輝度の有効ビットは，設定時に有効なビット数を表します．パレットが存在しない画面モードでは，この値は0となります．

表現に必要な最小値は，色／輝度を表現するときの数値上の最小値を表し，パレットが存在しない画面モードでは，この値は0となります．

取得バッファのデータ形式

最初の表示ページの情報

(DS：ECX)

0	B	青パレットの有効ビット
1	B	赤パレットの有効ビット
2	B	緑パレットの有効ビット
3	B	輝度の有効ビット
4	B	青色表現に必要な最小値
5	B	赤色表現に必要な最小値
6	B	緑色表現に必要な最小値
7	B	輝度表現に必要な最小値

ページ 1 の情報（画面総数＝2 のとき追加）

8	B	青パレットの有効ビット
		⋮

（ページ 0 と同様）

システム情報	1C0H
VRAM 有効ビットの取得	**機能コード 46H**

エントリ　AH　　　＝46H
　　　　　DX　　　＝解像度ハンドル
　　　　　DS：ECX ＝取得バッファのアドレス

リターン　EAX　　＝0（正常終了時）
　　　　　　　　　　－1（エラー）

説　明

ほかのオペレーションで参照した解像度ハンドルを使って，現在設定されている画面のVRAM有効ビットを取得するオペレーションです．現在表示されている画面のものを参照するときは，解像度ハンドルの指定を0とします．

取得バッファは，1ページ当たり4バイト必要です．もし，総ページ数がプログラミング時点で確定しない場合で，静的に割り付けするときは最大2ページ分用意するのが適当です．ただし，ページ0が表示不可能でページ1が表示可能な場合，バッファの先頭からページ1の内容が入ります．

「表示設定可能ページの取得（機能コード44H）」で，総ページ数を表示可能ページの合計により算出した結果によって動的に割り付けする場合は，そのページ数分のバイト数を確保します．

取得バッファのデータ形式

最初の表示ページの情報

(DS：ECX)

0	B	VRAMの青対応ビット数
1	B	VRAMの赤対応ビット数
2	B	VRAMの緑対応ビット数
3	B	VRAMの輝度対応ビット数

ページ1の情報（画面総数＝2のとき追加）

4	B	VRAMの青対応ビット数
		⋮

（ページ0と同様）

第17章

音源割り込み管理BIOS

音源割り込み管理BIOSは，マウス制御，音声処理，エンベロープ処理などを行うときのために，FM音源タイマの制御を行います．音源割り込みは頻度も高く，またサウンドとマウスの組み合わせにより制御が複雑化する傾向にあります．このため，第14章で述べた通常の割り込み管理を使用するとユーザープログラムに負担がかかりやすいことから，独立してサポートされることになったものです．

音源割り込み管理BIOSでは，ユーザー制御ルーチンを登録し，これらの割り込みが発生したとき，起動させることができます．

ここで対象となっている割り込みは，2系統に分類されています．ひとつはサウンド用，もうひとつはマウス用で，それぞれに適したBIOSが用意されています．

なお，付録C9に，音源割り込み管理BIOSを使ったサンプルプログラムが掲載されています．

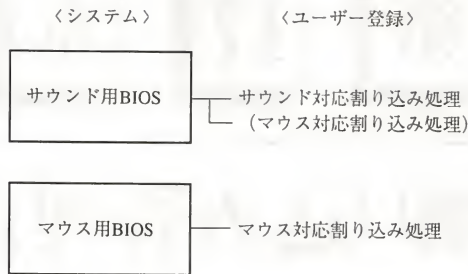
17.1 音源割り込み管理BIOSの概要

音源割り込みは，FM音源のタイマAおよびタイマBによる割り込みとPCM音源の割り込みを対象にしています．

●サウンド用BIOSとマウス用BIOS

サウンド用BIOSとマウス用BIOSの関係は，図のようになっています．

すなわち，一方のサウンド用BIOSは，サウンド対応割り込みとマウス対応割り込みに対処できます．他方，マウス用BIOSは，マウス対応割り込みにだけ対処できます．したがって，サウンド対応割り込みはサウンド用BIOSのもとでのみ働き，マウス対応割り込みは両方で作動することができます．

このような構造から，マウス対応割り込みとサウンド対応割り込みの登録が重なったときはサウンド用BIOSが使われ，マウス対応割り込みのみが登録されたときはマウス用BIOSが働きます．

サウンド用 BIOS は，具体的には次の処理を行います．

- タイマ A とタイマ B の再始動
- エンベロープ割り込みエントリ (50H) の処理
- 音声モード割り込みエントリ (51H) の処理
- マウス対応割り込みが登録されているときのマウス制御

また，マウス用 BIOS の処理内容は次のとおりです．

- タイマ B の再始動
- マウス制御

●タイマ B の作動間隔

サウンド用 BIOS では，タイマ B がエンベロープ処理とマウス制御に使われます．この場合，エンベロープ処理は 10ms 程度の間隔で行う必要があるので，時間間隔は通常 10.08ms(0DDH) に設定されています．

一方，マウスについては，この時間間隔では速過ぎるので，2 回に 1 回の割合で「間引き」してマウス対応割り込み処理を行わせ，CPU の負担を少なくしています．

マウス用 BIOS では，タイマ B はマウス専用に使えるので，時間間隔は 23.04ms(0B0H) に設定されています．

●割り出し処理

音源割り込み管理 BIOS では，割り込みのほかに，割り出しもサポートしています．

頻繁に発生する割り込みに対しては，対応処理に時間を割き過ぎると，続く割り込みに対応できなくなるため，割り込み処理の内容に限界があります．しかし，時には画面の描き換えなど時間がかかる処理をせざるを得ない場面があり，このようなときプログラムを工夫してカバーすることもできないことはありませんが，割り出しを使えば簡単です．

割り出しは，PIC を利用して，最初に割り込みが発生した段階で，優先度の低い割り込みにその処理を引き継がせます．こうすれば，低優先の処理をしながら，ほかの高優先の割り込みを受け付けることができ，例えば演奏を一時中断するなどの問題を避けることができます．

17.2　音源割り込み管理BIOS一覧

表 II–17–1 に，音源割り込み管理 BIOS の一覧を示します．

▼表 II-17-1　音源割り込み管理 BIOS 一覧

機能名称	機能コード
マウス対応割り込み 処理の登録	01H
マウス対応割り込み 処理の登録解除	02H
サウンド対応割り込み処理 の登録	03H
サウンド対応割り込み処理 の登録解除	04H
マウス割り込み動作回数の取得	05H
割り込み処理と割り出し処理の登録	06H
割り込み処理と割り出し処理の登録解除	07H
割り込み処理と割り出し処理の登録状態の取得	08H
マウス用／サウンド用各割り込み処理登録状態の取得	09H

17.3 音源割り込み管理BIOSリファレンス

音源割り込みBIOSについて，個別に詳しく説明します．

音源割り込み管理	1A0H
マウス対応割り込み処理の登録	機能コード01H

エントリ　AH　＝01H
DS：EDX　＝マウス割り込み処理用ローカルスタックのアドレス

リターン　EAX　＝0(サウンド用がすでに登録済)
－1(正常終了)

説　明　マウスの制御のための割り込み処理を登録します．このための割り込みにはタイマBが使われているので，それをサウンド用割り込みと併用するかどうかにより，動作が異なります．

1. 正常終了(戻り値＝−1)

- マウス用BIOSに設定，マウス割り込み処理登録
- マウス割り込み動作回数をクリア
- 割り込み処理用のローカルスタックアドレス設定
- FM音源初期化
- 割り込みコントローラの初期化

2. サウンド用BIOSがすでに登録されている場合(戻り値＝0)

- サウンド用BIOSのマウス割り込み処理登録
- マウス割り込み動作回数をクリア

音源割り込み管理	1A0H
マウス対応割り込み処理の登録解除	**機能コード 02H**

エントリ　AH　=02H

リターン　EAX　=0(サウンド用 BIOS が登録済)
　　　　　　　　−1(正常終了)

説　明

登録されているマウス対応割り込み処理を解除します．このための割り込みデバイスにはタイマBが使われているため，それをサウンド用割り込みと併用するかどうかで動作が異なります．

1. 正常終了(戻り値=−1)

- マウス用 BIOS と，マウス対応割り込み処理登録を解除
- FM 音源初期化
- 割り込みコントローラの初期化

2. サウンド用 BIOS がすでに登録されている場合(戻り値= 0)

- サウンド用 BIOS のマウス対応割り込み処理登録を解除

音源割り込み管理	1A0H
サウンド対応割り込み処理の登録	**機能コード 03H**

エントリ　AH　=03H
　　　　　DS：EDX　=サウンド割り込み処理用ローカルスタックのアドレス

リターン　EAX　=0(マウス用 BIOS の割り込み処理がすでに登録済)
　　　　　　　　−1(正常終了)

説　明

サウンド対応の割り込み処理を登録します．このための割り込みデバイスにはタイマBが使われているので，それをマウス用割り込みと併用するかどうかにより，動作が異なります．

1. 正常終了(戻り値＝−1)

- サウンド用BIOSに設定，サウンド対応割り込み処理登録
- 割り込み処理用のローカルスタックアドレス設定
- 割り込みコントローラの初期化

2. マウス用割り込みがすでに登録されている場合(戻り値＝0)

- マウス用BIOSをサウンド用BIOSに切り換え，サウンド対応割り込み処理登録
- サウンド用BIOSのマウス対応割り込み処理登録

音源割り込み管理　1A0H

サウンド対応割り込み処理の登録解除　機能コード04H

エントリ　AH　=04H

リターン　EAX　=0(マウス用が登録済)
　　−1(正常終了時)

説　明　登録されているサウンド対応用割り込み処理を解除します．このための割り込みにはタイマBが使われているため，それをマウス用割り込みと併用するかどうかで動作が異なります．

1. 正常終了(戻り値＝−1)

- サウンド用BIOSと，サウンド割り込み処理登録を解除
- 割り込みコントローラの初期化

2. マウス用割り込みがすでに登録されている場合(戻り値＝0)

- サウンド用BIOSをマウス用BIOSに切り換え
- マウス用BIOSのマウス割り込み処理に切り換え

音源割り込み管理	1A0H
マウス割り込み動作回数の取得	**機能コード 05H**

エントリ　AH　＝05H

リターン　EAX　＝マウス割り込み回数

説　明

マウス割り込みの動作を開始してから，現在までのマウス割り込み回数を EAX に取得します．

音源割り込み管理	1A0H
割り込み処理と割り出し処理の登録	**機能コード 06H**

エントリ　AH　＝06H

AL　＝割り込み (割り出し) 処理番号

0(タイマ A 割り込み処理 (EOI 前に呼び出し))

1(タイマ B 割り込み処理 (EOI 前に呼び出し))

2(タイマ A 割り込み処理 (EOI 後に呼び出し))

3(マウスイベント割り出し処理 1)

4(マウスイベント割り出し処理 2)

DS：ESI　＝パラメータ領域のアドレス

リターン　EAX　＝0(正常処理)

説　明

割り込み (割り出し) 処理のために呼び出されるルーチンのアドレスなどを，次のパラメータ形式で登録します．

(DS：ESI)

0	D	呼び出されるルーチンのアドレス
4	D	呼び出されるルーチンのセレクタ
n8	D	呼び出し時に設定される DS
12	D	呼び出し時に設定される ES
16	D	呼び出し時に設定される FS
20	D	呼び出し時に設定される GS

音源割り込み管理	1A0H
割り込み処理と割り出し処理の登録解除	機能コード07H

エントリ　AH　＝07H
　AL　＝割り込み(割り出し)処理番号
　　0(タイマA割り込み処理(EOI前に呼び出し))
　　1(タイマB割り込み処理(EOI前に呼び出し))
　　2(タイマA割り込み処理(EOI後に呼び出し))
　　3(マウスイベント割り出し処理1)
　　4(マウスイベント割り出し処理2)

リターン　EAX　＝0(正常処理)

説　明　割り込み(割り出し)処理のために登録したルーチンのアドレスなどを抹消して，対応処理を停止します．

音源割り込み管理	1A0H
割り込み処理と割り出し処理の登録状態の取得	機能コード08H

エントリ　AH　＝08H
　AL　＝割り込み(割り出し)処理番号
　　0(タイマA割り込み処理(EOI前に呼び出し))
　　1(タイマB割り込み処理(EOI前に呼び出し))
　　2(タイマA割り込み処理(EOI後に呼び出し))
　　3(マウスイベント割り出し処理1)
　　4(マウスイベント割り出し処理2)
　DS：ESI　＝パラメータ領域のアドレス

リターン　EAX　＝0(正常処理)

説　明　割り込み(割り出し)処理のために登録されているルーチンのアドレスなどを，次のパラメータ形式で参照します．

(DS：ESI)

オフセット		
0	D	呼び出されるルーチンのアドレス
4	D	呼び出されるルーチンのセレクタ
8	D	呼び出し時に設定される DS
12	D	呼び出し時に設定される ES
16	D	呼び出し時に設定される FS
20	D	呼び出し時に設定される GS

音源割り込み管理　1A0H

マウス用/サウンド用割り込み処理の登録状態の取得　機能コード 09H

エントリ　AH　＝09H

リターン　EAX　＝登録状態

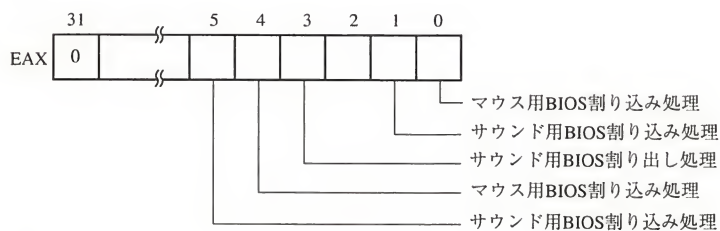
説　明

サウンド用 BIOS，マウス用 BIOS の割り込み処理登録の有無を，図の形式で参照します．ここでは，対応ビットの値により，

0：未登録

1：登録済

を表します．

第18章

MIDIマネージャBIOS

MIDI楽器の操作についてはサウンドBIOSにも関連オペレーションがありますが，MIDIマネージャBIOSは，それらを強化統合して拡張したものです．このBIOSは，TownsOS V2.1 L30からサポートされています．

通常のBIOSは，直接にはアセンブラで作成されたソフトから引用されますが，MIDIマネージャBIOSは一般にパラメータが多く，C言語によるソフトから間接的に引用するのに適しています．すなわち，ほかのBIOSとは異質なのです．

したがって，ここではCと関連させながら説明します．アセンブラでプログラミングする場合，Cの関係部分の説明も併せて参照してください．

18.1 MIDIマネージャBIOSの概要

MIDIマネージャBIOSでは，従来のEUPHONYドライバとMIDIエミュレータ，および標準MIDIファイル準拠フォーマットによる演奏の各機能がサポートされています．これらを使えば，FMTOWNSを480bpmの高分解能シーケンサとして駆動することができます．

ハードウェア対応については，従来のMIDI I/Fカード，FMT-40xシリーズのほかに，RolandのSuper-MPUまでも含まれます．

従来のアプリケーションでEUPHONYドライバを使用している場合は，ほとんど関数名を書き換えるだけでMIDIマネージャに置き換えることができます．

MIDIマネージャBIOSの主な仕様を図II-18-1に示します．

▼図II-18-1　MIDIマネージャBIOSの主な仕様

●最大トラック数　1〜256トラック(プログラマブル)
●最大同時発音数　任意(プログラマブル)
●テンポの範囲　5〜500
●MIDI入力フィルタ
　MIDIチャネル
　コントロールチェンジの置き換え
　ベロシティ固定
　トランスポーズ
●各トラックごとの制御情報
　出力ポート
　出力MIDIチャネル
　ミュート
　各ステータスごとのミュート
　トランスポーズ
　ベロシティのオフセット
　メインボリュームのオフセット
　エクスプレッションのオフセット
●コールバックルーチンの登録および呼び出し
　トラックのイベント出力毎
　入力ポートのイベント入力毎
　演奏の空き時間
●対応フォーマット
　標準MIDIファイル準拠フォーマット
　EUPフォーマット

18.2　MIDIマネージャBIOSの組み込み

MIDIマネージャは，TBIOSの一部として組み込まれます．使用するときは，TBIOS立ち上げのときに参照されるTBIOS.INIに，次の文を登録することによってメモリに常駐できます．

```
TMM=¥ TBIOS¥ MIDIMAN.BIN
```

18.3　MIDIマネージャBIOSの呼び出し

MIDIマネージャ(エントリアドレス110：0C0H)のシステムコールは，MIDIマネージャBIOSリファレンスで述べる機能コードや，エントリのために必要なレジスタおよびパラメータを設定したあと，次のように呼び出します．

```
push   fs
```

```
push  dword ptr 0110h
pop   fs
call  dword ptr fs:[0c0h]
pop   fs
```

18.4 MIDIとEUPHONYについて

ここでは，MIDIポートと，MIDIおよびEUPHONYのファイル形式について説明します．

18.4.1 MIDIポート

MIDIポートは，MIDIマネージャBIOSが入出力を行う対象です．ポートは複数使用するとき区別できるように，個別の番号が付けられています．そして，その番号系列には，物理ポートと論理ポートの2とおりがあります．

物理ポートは，MIDIインターフェースのハードウェアで決められた番号で，表II-18-1のように上位4ビットがインターフェースの種類，下位4ビットがその中の順番を示します．

論理ポートは，アプリケーション内で勝手に決めることができる仮想番号で，決定したら，BIOSに物理ポートとの対応づけを登録します．そうすると，あとは見かけ上論理ポートに対して入出力を行っているようにプログラミングできます．

以下の説明で，特にことわりのないときは,「ポート」とは論理ポートを指します．

▼表II-18-1　インターフェースタイプと物理ポート

タイプ別番号	I/F 種別	出力ポート数	入力ポート数
0nh	FMT-40x シリーズ	8	4
1nh	RS-MIDI	1	1
2nh	Roland Super-MPU	2	2
fnh	内蔵音源	1	0

18.4.2 標準MIDIファイル準拠フオーマット

標準MIDIファイルでは，具体的なコマンドに相当するイベントに，時間間隔を表すデルタタイムと，操作対象のトラック番号を加えた標識部を付けたデータを並べて，演奏情報を構成します(図II-18-2).

デルタタイムは，現在のイベントとひとつ前のイベントの時間間隔で，0ならばひとつ前のイベントと同時に処理することを意味します．それ以外では，指定された時間だけ経過した後，そのイベントが処理されます．

また，デルタタイムは可変長データで表現されます．具体的には，1バイトを7ビットだけ使用し，最上位1ビットにひとつ下のバイトと連結がある(1)かない(0)かのステータスの意味を持たせています．すなわち，6BH(2進で0110 1011)はそれ自身1バイトだけで完結していますが，85H/6BHは2進で1000 0101/0110 1011となり，ステータスを削除してシフトすると0000 0010/1110 1011(02H/EBH)が最終値として得られます．このように，固定長と違って，小さな数値なら少ないバイト数で表現できるのが特長で，まれに現れる大きな値のために最大桁だけ用意するような無駄を省く表現方法を実現しています．

イベントにはチャネルイベント(表II-18-2)，システムエクスクルーシーブイベント(表II-

▼図II-18-2 標準MIDIファイル準拠のフォーマット

標識部		コマンド	
デルタタイム	トラックNo.	イベント	演奏データ
デルタタイム	トラックNo.	イベント	
デルタタイム	トラックNo.	イベント	
⋮	⋮	⋮	
デルタタイム	トラックNo.（0）	イベント（終端マーカ）	

▼表II-18-2 標準MIDIファイル準拠フォーマット(チャネルイベント)

イベント(フォーマット)	説　明
ノートオフ (0x8n, Note, Velocity)	ノートオンとノートオフを別々に管理したい場合に用いる．その場合は対応するノートオンのDurationを0にする．
ノートオン (0x9n, Note, Velocity, Duration(VLen)[, Off-velocity])	ノート番号(Note)，ベロシティ(Velocity)に続いてそのノートの長さ(Duration)，さらに対応するノートオフのオフベロシティ(Off-velocity)が並ぶ．長さは可変長データで表す．もし，長さが0のときは対応するノートオフを出力しないのでオフベロシティは不要．
ポリフォニックキープレッシャ (0xan, Note, Value)	ノート番号と圧力値を指定する．
コントロールチェンジ (0xbn, Number, Value)	コントロール番号とコントロール値を指定する．
プログラムチェンジ (0xcn, Number)	プログラム番号を指定する．
チャネルプレッシャ (0xdn, Value)	圧力値を指定する．
ピッチベンド (0xen, ValueL, ValueH)	ベンド値(上位，下位7ビットずつの計14ビット)を指定する．

すべてのチャネルメッセージはMIDIチャネルを持っていて，実際に出力するときは，そのトラックの出力MIDIチャネルに置き換えられる．ただし，そのトラックの出力MIDIチャネルが〝omni〃であるときには，そのままのMIDIチャネルで出力される．また，標準MIDIファイルではランニングステータスによるステータスの省略が許されるが，MIDIマネージャではステータスは省略せずに必ず入れること．

18–3)，メタイベント(表II–18–4)があり，それぞれ表中に示すような働きとフォーマットを持っています．

▼表II-18-3　標準MIDIファイル準拠フォーマット(システムエクスクルーシブイベント)

イベント(フォーマット)	説　明
システムエクスクルーシブメッセージ (0xf0, Length(VLen), data....)	0xf0を除くエクスクルーシブデータの長さを可変長で入れ，その後にデータの本体を置く．実際には0xf0が出力され，それに続いて指定された長さのデータを送る．必ずしも0xf7(eox)で終わる必要はないが，その場合は，次の継続データを使ってエクスクルーシブを完結させること．
継続エクスクルーシブ (0xf7, Length(Vlen), data....)	エクスクルーシブデータが1つのパケットに収まらない場合などに使用する．最初の0xf7は出力されない．

▼表II-18-4　標準MIDIファイル準拠フォーマット(メタイベント)

イベント(フォーマット)	説　明
テンポ 0xff, 0x51, 0x03, tempo	tempoは3バイトで四分音符の長さ(単位μsec)を表す．例えば1分当たり四分音符で120拍(120bpm)なら60000000μsec/120beatで500000μsecが実際の値となる．30bpmなら60000000μsec/30beatで2000000μsecが実際の値に対応する．このイベントがない場合は120bpmで演奏される．
拍子記号 0xff, 0x58, 0x04, signature	signatureは4バイトで，第1バイト目が拍数を，第2バイト目が分母を表す．第3バイト目はメトロノームの間隔(MIDIクロック数)を，第4バイト目は音符の記譜上の長さが入る．MIDIマネージャでは第3，第4バイト目は無視される．第2バイト目の分母は，2のマイナス乗が入る．つまりn/4なら2がn/8なら3が入る．このイベントがない場合は4/4で演奏される．小節の途中の中途半端な位置にあった場合は，それよりあとの演奏がうまくできない恐れがあるので，拍子記号は必ず正しい位置に入れること． 例) 0xff, 0x58, 0x04, 0x04, 0x02, 0x18, 0x08 → 4/4 0xff, 0x58, 0x04, 0x02, 0x02, 0x18, 0x08 → 2/4 0xff, 0x58, 0x04, 0x03, 0x03, 0x0c, 0x08 → 3/8 0xff, 0x58, 0x04, 0x08, 0x04, 0x06, 0x08 → 8/16
終端マーカー 0xff, 0x2f, 0x00	曲の最後には必ずこのマーカーを入れる．
データ継続 0xff, 0x7f, 0x01, 0x4b	曲データが継続することを意味する．リングバッファなどにデータを転送しながら演奏する場合などに，MIDIマネージャが書き込みポインタを追い越してしまわないようにするためにある．

メタイベントは直接MIDI出力されるデータではないが，シーケンス名やテキスト，歌詞などのデータがあり，特に進行上非常に重要なテンポや拍子も含まれる．メタイベントは0xffで始まり，その次の1バイトがそれがどのようなメタイベントであるかを表す．さらに，可変長でそのデータ本来の長さが置かれ，その直後にデータ本体が続く．MIDIマネージャでは，以下のメタイベント以外は無視する．このメタイベントはトラックに関係なく処理されるので，どのようなトラック番号が入っていてもよい．

18.4.3 EUPファイルフォーマット

従来からのFMTOWNS用演奏フォーマットで，タイムベースは四分音譜1個について96クロックです．データは6バイトを1パケットとする固定長で，図II–18–3のような様式が決め

られています.

イベントにはチャネルイベント(表II-18-5), システムエクスクルーシーブイベント(表II-18-6), およびその他(表II-18-7)があり, それぞれ表中に示すような働きとフォーマットを持っています.

▼図II-18-3 EUPファイルのパケット

+0	+1	+2	+3	+4	+5
ステータス	トラック	タイム (Low)	タイム (High)	データ1	データ2

▼表II-18-5 EUPフォーマット(チャネルイベント)

イベント	フォーマット						説明
	ステータス (1バイト)	2バイト	3バイト	4バイト	5バイト	6バイト	
ノートオフ	$8n	Duration				OFF	ノートの長さ(Duration)は下位4ビットずつを取り出して16ビットにする.
		LSB L	LSB H	MSB L	MSB H	VELO	
ノートオン	$9n	TRACK 番号	TIME LSB	TIME MSB	音程	ON VELO	ノートオンは必ず対となるノートオフが直後に来る.
ポリフォニック キープレッシャ	$An	TRACK 番号	TIME LSB	TIME MSB	音程 (NOTE)	プレッシャー	ノート番号と圧力値を指定する.
コントロール チェンジ	$Bn	TRACK 番号	TIME LSB	TIME MSB	CONTROL 値	設定値	コントロール番号とコントロール値を指定する.
プログラム チェンジ	$Cn	TRACK 番号	TIME LSB	TIME MSB	PROGRAM 値	ダミー	プログラム番号を指定する.
チャネル プレッシャ	$Dn	TRACK 番号	TIME LSB	TIME MSB	PRESSER 値	ダミー	圧力値を指定する.
ピッチベンド	$En	TRACK 番号	TIME LSB	TIME MSB	BEND 値 LSB	BEND 値 MSB	ベンド値(上位, 下位7ビットずつの計14ビット)を指定する.

すべてのチャネルメッセージはMIDIチャネルを持っている. そして, 実際に出力するときは, そのトラックのMIDIチャネルに置き換えられる. ただし, そのトラックのMIDIチャネルが〝omni〞であるときには, そのままのMIDIチャネルで出力される.

▼表II-18-6 EUPフォーマット(システムエクスクルーシブイベント)

イベント	フォーマット						説明
	ステータス (1バイト)	2バイト	3バイト	4バイト	5バイト	6バイト	
エクスクルーシブ ステータス	$F0	TRACK 番号	TIME LSB	TIME MSB	ダミー ($FF)	ダミー ($FF)	エクスクルーシブデータの最初のパケット
データ	データ列 不定バイト数(6バイト単位)						6の倍数バイトで任意のデータが並ぶ(余った部分は$FFで埋める)
END OF エクスクルーシブ	$F7	ダミー ($FF)	ダミー ($FF)	ダミー ($FF)	ダミー ($FF)	ダミー ($FF)	エクスクルーシブデータの最後のパケット

エクスクルーシブデータの最初のパケットにはデータは入らず, 2番目のパケットからデータが入る. このエクスクルーシブの間には他のパケットが入ってはならず, また, 最後は必ず$F7(End of Exlusve)の含まれるパケットで終わらなければならない. その場合パケット内の余分なところには$FFを入れる.

▼表 II-18-7　EUP フォーマット (その他のイベント)

イベント	フォーマット						説　明
	ステータス (1 バイト)	2 バイト	3 バイト	4 バイト	5 バイト	6 バイト	
小節マーカー	$F2	拍子値	TIME LSB	TIME MSB	ダミー	ダミー	重要なイベントで，EUP フォーマットでは 1 小節毎に必ずこの小節マーカーがなければならない．TimeL, TimeH はこの小節マーカーまでの時間，つまり前の小節の長さが入る．signature にはこの小節の拍子が入る．この小節マーカーの TimeL, TimeH と前の小節マーカーの signature が矛盾しないことが重要で，例えば，前の小節マーカーの signature が 4/4 なら，この小節マーカーの Time は 384(96＊4) でなければならない．
テンポ	$F8	ダミー	TIME LSB	TIME MSB	テンポ LSB	テンポ MSB	テンポは 14 ビットで表現する．取りうる範囲は 0～250 までで，実際の値はそれに 30 を足した，30bpm～280bpm となる．
USER CALL PROGRAM (NOT SUPPORT)	$FA	TRACK 番号	TIME LSB	TIME MSB	PROGR-AM 値	ダミー	未サポート
パターン番号 (NOT SUPPORT)	$FB	TRACK 番号	TIME LSB 0	TIME MSB 0	パターン番号	ダミー	未サポート
TRACK COMMAND	$FC	TRACK 番号	TIME LSB 0	TIME MSB 0	コマンド	データ (変更値)	演奏中にポートや MIDI チャネルを変更する．コマンド＝1: ポート，2:チャネル
DATA CONTINUE	$FD	—	—	—	—	—	曲データが継続することを意味する．リングバッファなどにデータを転送しながら演奏する場合などに，MIDI マネージャが書き込みポインタを追い越してしまわないようにするために利用する．
終端マーカー	$FE	拍子値	TIME LSB	TIME MSB	ダミー	ダミー	曲の最後には必ずこのマーカーを入れる．TimeL, TimeH はこの小節マーカー同様にこの終端マーカーまでの時間，つまり最後の小節の長さが入る．
ダミーコード	$FF						何もしないイベントとして扱われる (無視される)．

18.5 MIDIマネージャ関連Cソースライブラリ定義

MIDIマネージャ用のソースライブラリ〝mmDEF.H″に次の定義があります.

18.5.1 型宣言

構造体を除き,図II-18-4の型宣言がなされています.

▼図II-18-4 構造体以外の型宣言

```
typedef unsigned int    SIGN;     /* 拍子記号          */
typedef int             TIME;     /* 演奏位置(クロック) */
typedef unsigned char   TRACK;    /* トラック番号      */
typedef unsigned char   PORT;     /* MIDIポート番号    */
typedef unsigned char   MIDICH;   /* MIDIチャンネル    */
typedef unsigned char   FLAG;     /* フラグ            */
```

18.5.2 構造体

BIOSで使われるパラメータは,構造体で引き渡しされるものが少なくありません.したがって,アセンブラだけでプログラミングする場合も,構造体の内容を知っておく必要があります.

● TRACKWORK構造体

個別のトラックの制御情報を記述する構造体です.アプリケーションでは,出力ポートなどのメンバを書き換えることによって,期待する値に設定します.

▼図II-18-5 TRACKWORK構造体

```
typedef struct
{
        TRACK   trk_number;     /*track number          */
        FLAG    trk_mute;       /*play or mute          */
        int     trk_filter;     /*play filter bits      */
        PORT    trk_port;       /*output port           */
        MIDICH  trk_midich;     /*output midi ch        */
        char    trk_notemap;    /*note replace map      */
        char    trk_ctrlmap;    /*ctrl replace map      */
        char    trk_progmap;    /*prog replace map      */
        char    trk_bias;       /*velocity shift        */
        char    trk_oct;        /*transpose shift       */
        char    trk_vol;        /*ctrl volume shift     */
        char    trk_exp;        /*ctrl expression shift */
        char    trk_resv1;      /*reserve               */
}TRACKWORK;
```

```
TRACK trk_number;
```

トラックの番号を入れる．ここは内部で構造体アドレスからトラック番号を得るためのフィールドなので，必ず0から順に番号を与えること．

```
FLAG trk_mute;
```

そのトラックのすべての出力をまとめてon/offする．0以外なら出力し，0なら停止する．

```
int trk_filter;
```

各種MIDIメッセージを出力するかどうかを決めるビットフィールド．ビットの対応は以下のとおりで，ビットの立っているメッセージは出力される．通常はすべてonにしておく．

```
#define FLTNOTEOFF                        0x01
#define FLTNOTEON                         0x02
#define FLTKEYPRESS                       0x04
#define FLTCTRL                           0x08
#define FLTPROG                           0x10
#define FLTCHPRESS                        0x20
#define FLTPITCH                          0x40
#define FLTEXC                            0x80
#define FLTTEMPO                          0x100
#define FLTMETA                           0x200
```

```
PORT trk_port;
```

トラックのデータ，出力ポートを指定する．

```
MIDICH trk_midich;
```

トラックの出力MIDIチャネルを指定する．ここで0〜15を選択すればそのトラックの演奏データはすべてそのチャネルに置換されて出力される．0xff(omni)を指定すれば，演奏データが持っている個々のチャネルでそのまま出力される．

```
char trk_notemap;
char trk_ctrlmap;
char trk_progmap;
```

このトラックから出力されるノートやコントロールチェンジ，プログラムチェンジの番号の置き換えやマスクを行う時に指定し，0〜15までのどのマップを使用するかを選択する．置き換え等を行いたくない場合は −1 を設定する．

```
char trk_bias;
char trk_oct;
char trk_vol;
char trk_exp;
```

trk_biasはトラックから出力されるnote onのベロシティに，trk_octはnote on/note off/polyphonic key press等の音程情報にこの値を加える．trk_vol，trk_expはcontrol changeのmain volumeとexpressionにこの値を加える．トラック全体の音量等を変化させたいときに使用する．

● NOTEOFFTABLE構造体

MIDIマネージャが演奏時のnote on/offを管理するテーブルで，後述のMIDIMANCTRL構造体を経由して参照されます．実際のテーブルは，この定義のサイズ×(最大同時発音数+1)だけとられ，演奏を行うと，note on/offにより各種の値が書き込まれます．

0番目の配列のoff_timeには，現在発音中の音数が入ります．残りの部分では，off_timeはそのノートがoffになる時点を表し，off_flagが1のところに対応する部分は現在発音中であることを示します．off_trackは，そのノートがどのトラックのものかを表します．

▼図II-18-6　NOTEOFFTABLE構造体

```
typedef struct
{
        int     off_time;
        FLAG    off_flag;
        TRACK   off_track;
        PORT    off_port;
        MIDICH  off_ch;
        char    off_note;
        char    off_velo;
}NOTEOFFTABLE;
```

● REALTIME構造体

実時間を表す構造体です．64ビットで構成され，1/480000000秒を単位として，時間を表します．

▼図II-18-7　REALTIME構造体

```
typedef struct
{
        int     rt_lo;
        int     rt_hi;
} REALTIME;
```

● PLACE構造体

演奏位置を表す構造体で，内容は，小節，拍，チック，クロック数，実時間，拍子記号で構成されます．

これらの項目は，いわば五線譜上の座標位置のようなもので，演奏開始ポイントをドライバに通知したり，現在の演奏位置を知るのに用いられます．

▼図 II-18-8　PLACE 構造体

```
typedef struct
{
        int             p_meas;
        int             p_beat;
        int             p_tick;
        TIME            p_clock;
        REALTIME        p_time;
        SIGN            p_sign;
} PLACE;
```

● SMPTE 構造体

Roland 社の Super-MPU の SMPTE READ/WRITE 機能を利用するときのための構造体で，タイムコードを表します．内容は時，分，秒，フレーム，ビットで構成され，フレームの扱いにより 4 種類のタイプがあります．

フレーム名の中の数値は 1 秒のフレーム数を表し，例えば SMPTE30 ならば 30 フレームで 1 秒となりますが，実際の NTSC 方式では 29.97 であり，分に直すと 3.6 秒の誤差が出ます．そこで，これに対応するため，SMPTE30DF では最初の 2 フレームをスキップして，誤差を 60ms 以内に抑えています．

SMPTEREAL は，計算しやすいように，100 フレームで 1 秒としたものです．

▼図 II-18-9　SMPTE 構造体 (下) とフレーム数シンボル定義 (上)

```
#define SMPTE24         (0)
#define SMPTE25         (1)
#define SMPTE30DF       (2)
#define SMPTE30         (3)
#define SMPTEREAL       (4)

typedef struct
{
        int             tc_hr;
        int             tc_min;
        int             tc_sec;
        int             tc_fr;
        int             tc_bit;
        int             tc_type;
} SMPTE;
```

● METRONOME 構造体

演奏中にメトロノームを鳴らすときに使われる構造体です.

▼図 II-18-10　METRONOME 構造体

```
typedef struct
{
        unsigned char   metro_mode;
        PORT            metro_port;
        MIDICH          metro_midich;
        char            metro_hinote;
        char            metro_hivelo;
        char            metro_lownote;
        char            metro_lowvelo;
        char            metro_resv;
        TIME            metro_duration;
} METRONOME;
```

```
unsigned char metro_mode;
```

メトロノームを鳴らすかどうかを決める. 0 なら鳴らさず, 1 なら録音中のみ鳴らす. 2 なら録音中, 再生中ともに鳴らす..

```
PORT            metro_port;
MIDICH          metro_midich;
```

メトロノームを出力するポートと MIDI チャネル.

```
char            metro_hinote;
char            metro_hivelo;
char            metro_lownote;
char            metro_lowvelo;
char            metro_resv;
```

メトロノームの強拍, 弱拍それぞれのノート番号およびベロシティ.

```
TIME            metro_duration;
```

メトロノームの各拍の鳴っている時間.

● MIDIMANCTRL 構造体

MIDI マネージャをオープンするときに使われる構造体です. 必要なバッファの登録など準備手続きをするのに利用します.

▼図 II-18-11 MIDIMANCTRL 構造体

```
typedef struct
{
        int             mc_interval;
        int             mc_maxnote;
        int             mc_maxtrack;
        unsigned char   *mc_sbios;
        NOTEOFFTABLE    *mc_noteoff;
        TRACKWORK       *mc_trackwork;
        char            *mc_filter[16];
        unsigned char   *mc_event;
        int             mc_eventsize;
        unsigned char   *mc_fmtout;
        int             mc_fmtoutsize;
        unsigned char   *mc_fmtin;
        int             mc_fmtinsize;
        unsigned char   *mc_smpu;
        int             mc_smpusize;
        unsigned char   *mc_stackadr;
        int             mc_stacksize;
} MIDIMANCTRL;
```

`int mc_interval;`

FMタイマおよびMIDIカード(FMT-401/402/403)のPIT(プログラマブルインターバルタイマ)の割り込み間隔を1/480000000秒単位(480000で1msec)で指定する．

この間隔は演奏時のシーケンサの分解能に影響する．例えばテンポが十分に速いと1クロックの間隔が極端に小さくなるが，実際に人間の耳で聞き分けられる間隔以下になり，そのままではコンピュータの処理に大きな負担がかかる．そこで，ある間隔より短い場合は一括して処理を行えば，聴感上は違和感なくかつコンピュータにかかる負担を小さくすることができる．つまり，2msecを指定すればテンポに関係なく最大2msec以内の誤差でイベントが処理される．

通常のシーケンサでは2msecから5msec位，曲のノリをそれほどシビアに必要とせずかつ処理の負担を軽くしたいアプリケーション(ゲームソフトなど)であれば5msecから20msecあたりが適当．また，vsync割り込みなどを使用して自分で割り込み処理エントリをコールする場合はこの値を0とすること．その場合，ドライバは自分の割り込みを登録しない．

`int mc_maxnote;`

全トラックで同時に発音する音の数を指定する．シーケンスソフトのような同時発音数が不足となる場合は十分に大きな値，例えば256位を指定するとよい．

この値は後に出てくるノート管理テーブル(mc_noteoff)のサイズに影響する．

`int mc_maxtrack;`

最大トラック数を指定する．例えばコンダクタトラック＋99データトラックのシーケンサであるなら100を指定する．内蔵音源しかドライブしないものや，1つのMIDIポートしか扱わないようなものであるならもっと小さくしてもよい．

この値は後に出てくるトラック管理テーブル(mc_trackwork)のサイズに影響する．

`unsigned char *mc_sbios;`

初期化されたサウンドBIOSのワークアドレス．

```
NOTEOFFTABLE *mc_noteoff;
```

演奏時のノートオン・オフを管理するためのワーク．このワークのサイズは最大同時発音数(mc_maxnote)×(NOTEOFFTABLE構造体+1)で求められる．

```
例)    NOTEOFFTABLE noteoff_buff[MAXPLAYNOTE+1];
```

```
TRACKWORK *mc_trackwork;
```

各トラックの管理情報を記憶するためのワーク．このワークのサイズは最大トラック数(mc_maxtrack)×TRACKWORK構造体のサイズで求められる．

```
例)    TRACKWORK trackwork_buff[MAXTRACK];
```

```
char *mc_filter[16];
```

プログラムチェンジやコントロールチェンジの入力時の置換フィルタを登録する．1つのフィルタは，128Byteからなり，そのテーブルの値を参照してコントロールチェンジを違う番号に置き換えたり，負の値にしてマスクしたりできる．

フィルタの器は16個まで登録できるが，フィルタを使用しない場合はNULLを入れる．実際にどのフィルタをどのメッセージに適用するかは，assign mapを通して参照される．

```
unsigned char *mc_event;
int mc_eventsize;
```

各ポートからのMIDIイベントの入出力の際の使用するバッファを設定する．通常のMIDIメッセージは2または3Byteなので，それに時間情報(4Byte)とサイズ(4Byte)を足しても16Byteもあれば足りる．しかし，Exclusiveデータを扱う場合にはこのバッファを十分に大きく取らないとデータが細切れになってしまうので，高機能なシーケンスソフトなどでは，できればデータの細切れは避けたい．

Exclusiveデータを受信した場合，このバッファに入りきらないときは続きのパケットとなる．バルクダンプなどを除けば大抵のExclusiveデータは256Byteに収まるので300Byte以上もあれば十分と言える．

```
unsigned char *mc_fmtout;
int mc_fmtoutsize;
unsigned char *mc_fmtin;
int mc_fmtinsize;
unsigned char *mc_smpu;
int mc_smpusize;
```

FMT-40xシリーズ用の入出力バッファおよびS-MPU用の入力バッファを設定する．FMT-40xシリーズ用は最大4枚まで実装できるため最大入力4ポート，最大出力8ポートとなるので，十分に大きな値を設定すること．標準的には出力側64×8Byte，入力側128×4Byte程度．

```
例)    char fmt_out_buff[64*8];
       char fmt_in_buff[128*4];
       char smpu_in_buff[128];

       midiman.mc_fmtout = fmt_out_buff;
       midiman.mc_fmtoutsize = 64*8;
       midiman.mc_fmtin = fmt_in_buff;
       midiman.mc_fmtinsize = 128*4;
       midiman.mc_smpu = smpu_in_buff;
       midiman.mc_smpusize = 128;
```

```
unsigned char *mc_stackadr;
int  mc_stacksize;
```

割り込みルーチンで使用されるローカルスタックを設定する．割り込みはレベル4，5およびその割り出しのための合計4個のエリアが必要．内部では，ここで指定したバッファを4分割して使用するので，十分に大きなサイズを確保しないとスタックオーバーフローとなる．推奨は768×4Byte以上で，次に述べる各種ユーザールーチンの中でスタックを多く消費する可能性があるなら，さらにその分を加える．

例)
```
char local_stack[1024*4];

midiman.mc_stackadr = local_stack;
midiman.mc_stacksize = 1024*4;
```

● FUNCCTRL構造体

アプリケーションで，割り込みルーチンから呼び出されるコールバックルーチンを登録するとき，およびその内容を参照するときに使われる構造体です．コールバックルーチンは，同時に最大4組(4つのアプリケーション)に対応する内容を登録することができます．

各々の1組については，演奏データ作成や録音の前処理，イベントの監視などの手続きを登録します．必要がないエントリは，その関数のポインタをNULLに設定すればバイパスされます．付け加えるならば，これら3つのエントリは，FARコールで呼び出されます．

エントリで指定された関数では，レジスタの値を変更するような処理をすることは禁止されています．すなわち，アセンブラでは，使用するレジスタについて入口で退避し，出口で復帰させる処理を記述しなければなりません．

C言語では，このような機能がないので，必要なときはライブラリのTMM_setCFunc()関数を利用します．この関数は，間に入って,「ユーザーコールバックルーチンの登録(機能コード07H)」を実行してくれます．参考までに，解除はTMM_resetUserFunc()によります．

ユーザーコールバックルーチンは割り込み処理の一部であり，スタックの容量が限られていること，そこからDOSの呼び出しはできないことに注意してください．

▼図II-18-12　FUNCCTRL構造体

```
typedef struct
{
        void            (*(fc_idletask))();
        short int       fc_idleseg;
        void            (*(fc_rectask))();
        short int       fc_recseg;
        void            (*(fc_eventtask))();
        short int       fc_eventseg;
        short int       fc_dataseg;
}FUNCCTRL;

typedef struct
{
        void            (*(cfc_idletask))();
        void            (*(cfc_rectask))(PORT,FLAG,TRACK,unsigned char *);
        void            (*(cfc_eventtask))(TRACK,PORT,unsigned char *);
}CFUNCCTRL;
```

```
void            (*(fc_idletask))();
short int       fc_idleseg;
```

演奏やMIDIの入出力等の動作がない空き時間に呼び出されるコールバックルーチンで，演奏時にデータをインタプリトしながら作成していくようなアプリではここにそのルーチンを登録することによって，あたかも内部データをそのままドライブするように動かすことができる．

```
void            (*(fc_rectask))();
short int       fc_recseg;
```

入力ポートに何かMIDIイベントが到着すると呼び出されるコールバックルーチンで，すでに演奏時の時間が付加されているのでリアルタイム録音などに利用できる．このルーチンが呼び出された時，ds：ediの指すアドレスにイベントが格納されている．先頭のdwordに演奏クロック，次のdwordにそのイベントのサイズ，その次，つまり8Byte目から実際のイベントが格納されている．また，bhには演奏，一次停止，録音の状態，blレジスタにはそのイベントの入力ポート，dhレジスタにはそのイベントがassign filterによって出力するかどうか(0以外なら出力)の情報，dlレジスタには出力する際の出力フィルタのトラック番号が入っている．したがって，入力情報をassign outでMIDIポートに出力する前にデータを書き換えることもできる．

```
void            (*(fc_eventtask))();
short int       fc_eventseg;
```

演奏時に出力ポートからデータが出力されるたびに呼び出されるコールバックルーチンで，演奏情報を画面に出力したい場合などに利用できる．ただし注意しなければならないのは，このルーチンは割り込み処理の一部であるということ．ここでダイレクトに画面描き換え等の重い処理をドライブすると，演奏がよれたり思いがけないバグを発生させることになるので，ここのルーチンではイベントをキューに積むなどの最小限の処理に抑えておかなければならない．このルーチンが呼び出された時，ds：ediの指すアドレスにイベントが，また，blレジスタにはそのイベントの出力ポートが，bhレジスタにはそのイベントがどのトラックのものかが入っている．したがって，実際にデータがMIDIポートに出力される前にデータを書き換えることもできる．

```
short int       fc_dataseg;
```

各コールバックルーチンの呼び出し時にDS，ESセグメントに設定する値．

● RSCTRL構造体

MIDIマネージャはRS-MIDIのI/Oを行うドライバを持っていません．I/Oをコントロールするモジュールはライブラリ等からTMM_initRsMidiを通して登録され，MIDI Managerからコールバックされます．その入出力処理ルーチンのエントリを示すためにRSCTRL構造体を使用します．この構造体で登録されたユーザールーチンは，FARコールによりコールバックされます．

出力／入力／データについて，セグメントとオフセット(データを除く)を記述します．

▼図II-18-13　RSCTRL構造体

```
typedef struct
{
        void            (*(rc_output))();
        short int       rc_outputseg;
        void            (*(rc_input))();
        short int       rc_inputseg;
        short int       rc_dataseg;
}RSCTRL;
```

● ASSIGNFILTER 構造体

MIDI 入力の取り扱いを規定する構造体で，4 つまで登録できます．そして，各ポートごとに，どの ASSIGNFILTER を使うか選択できます．

あるポートにイベントが到着すると，該当する ASSIGNFILTER 構造体の as_status と as_chmap を参照して，入力すべきものかどうかを確認します．適当と判断されたら，ベロシティ補正などの処理をして，ユーザーの録音ルーチンを呼び出します．続いて，その結果を指定トラックの出力パスに変換して MIDI に出力します．

▼図 II-18-14　ASSIGNFILTER 構造体

```
typedef struct
{
        short int       as_status;
        short int       as_chmap;
        TRACK           as_track;
        char            as_velo;
        char            as_oct;
        char            as_ctrlmap;
} ASSIGNFILTER;
```

```
short int       as_status;
```

各ビットがステータスに対応する (FLTNOTEOFF〜FLTMETA の BitON で入力)

```
short int       as_chmap;
```

bit0〜15 がそれぞれの MIDI チャネル 1〜16 に対応

```
TRACK           as_track;
```

出力トラック

```
char            as_velo;
```

0 なら置き換えなし

```
char            as_oct;
```

ノートナンバーに対するオフセット

```
char            as_ctrlmap;
```

コントロールチェンジを置き換えるマップナンバー

18.6　MIDIマネージャBIOS一覧

表II-18-8に，MIDIマネージャBIOSの一覧を示します．

▼表II-18-8　MIDIマネージャBIOSの機能一覧

機能名称	機能コード
MIDIマネージャのオープン	00H
MIDIマネージャのクローズ	01H
MIDIMANCTRL情報の取得	02H
RS-MIDIルーチンの登録	03H
RS-MIDIルーチンの解除	04H
割り込み処理用エントリ	05H
割り出し処理用エントリ	06H
ユーザーコールバックルーチンの登録	07H
ユーザーコールバックルーチンの取得	08H
ユーザーコールバックルーチンの解除	09H
MIDIデータの出力	0AH
演奏の開始	10H
演奏の終了	11H
演奏の一時中断	12H
演奏の再開	13H
演奏モードの設定	14H
演奏位置の取得	16H
テンポの設定	17H
テンポの取得	18H
相対テンポの設定	19H
相対テンポの取得	1AH
EUPデータ相対テンポの設定	1BH
EUPデータ相対テンポの取得	1CH
ステップモードの進行	1DH
同期モードの設定	20H
SMPTE開始位置の設定	21H
SMPTE同期精度の設定	22H
S-MPU内部時間の設定	23H
実時間からSMPTE時間への変換	24H
SMPTE時間から実時間への変換	25H
リモートモードの設定	26H
同期信号出力の設定	27H
メトロノームの設定	28H
アサインマップの設定	30H
アサインマップの取得	31H
アサインフィルタの設定	32H
アサインフィルタの取得	33H
出力ポートマップの設定	34H
出力ポートマップの取得	35H
入力ポートマップの設定	36H
入力ポートマップの取得	37H
内蔵音源の初期化	40H
内蔵音源へのMIDIデータ出力	41H
内蔵音源のMIDIチャネルの設定	42H
内蔵音源のMIDIチャネルの取得	43H
内蔵音源のマスターボリュームの設定	44H
内蔵音源のマスターボリュームの取得	45H

18.7　MIDIマネージャBIOSリファレンス

MIDIマネージャBIOSについて，個別に詳しく説明します．

MIDIマネージャ	COH
MIDIマネージャのオープン	機能コード00H

エントリ　AH　=00H
DS：ESI　=MIDIMANCTRL構造体へのポインタ

リターン　EAX　=0(正常終了時)
−3(バッファ，スタックなどの容量不足)
−13(ベクタ設定時のエラー)
−14(すでにオープン済)

説　明　MIDIマネージャドライバの初期化を行います．パラメータは次の形式で指定します．

(DS：ESI)

0	struct	MIDIMANCTRL

該当C関数　int TMM_openMidiMan(MIDIMANCTRL *midimanctrl);

MIDIマネージャ	COH
MIDIマネージャのクローズ	機能コード01H

エントリ　AH　=01H

リターン　EAX　=0(正常終了)

説　明　MIDIマネージャドライバの終了処理を行います．

該当C関数　int TMM_closeMidiMan();

MIDI マネージャ	COH
MIDIMANCTRL 情報の取得	機能コード 02H

エントリ　AH　=02H
DS : ESI　=MIDIMANCTRL 構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　現在設定されている MIDIMANCTRL 情報を読み出し，次の形式で構造体に収容します．

(DS : ESI)

0	struct	MIDIMANCTRL

該当C関数　int TMM_getMidiManCtrl(MIDIMANCTRL *midimanctrl);

MIDI マネージャ	COH
RS-MIDI ルーチンの登録	機能コード 03H

エントリ　AH　=03H
DS : EBX　=RSCTRL 構造体へのポインタ

リターン　EAX　=0(正常終了)
-14(すでに登録済)

説　明　RS-232C MIDI の入出力コールバックルーチンを登録するオペレーションです．パラメータは，次の形式で定義します．

(DS : EBX)

0	struct	RSCTRL

該当C関数　int TMM_initRsMidi(RSCTRL *rsctrl);

MIDIマネージャ	C0H
RS-MIDIルーチンの解除	**機能コード04H**

エントリ　AH　=04H

リターン　EAX　=0(正常終了)

説　明　RS-232C MIDIの入出力コールバックルーチンの登録を解除するオペレーションです.

該当C関数　int TMM_termRsMidi();

MIDIマネージャ	C0H
割り込み処理用エントリ	**機能コード05H**

エントリ　AH　=05H

EDX　=前回呼び出しからの経過時間(1/480μs単位)

リターン　なし

説　明　外部割り込みを使ってMIDIドライバを作動させるための割り込みエントリです.

MIDIMANCTRLのmc_intervalに0を指定したとき，ユーザーが任意の割り込みを使用して，独自の時間間隔を作り出すことができます．したがって，そのタイミングになったら，ユーザーは割り込み処理の中でこのBIOSを使用して，MIDIマネージャに前回呼び出しからの経過時間(1/480μs単位)を知らせます.

MIDIマネージャは，このエントリが呼び出されると，内部の時間を更新して直ちにリターンします．続いて，アプリケーションの側では，PICにeoiを送り，次に述べる割り出し処理エントリを呼び出します.

該当C関数　なし

MIDIマネージャ	C0H
割り出し処理用エントリ	機能コード06H

エントリ　AH　=06H

リターン　なし

説明　外部割り込みを使ってMIDIドライバを作動させたときための割り出しエントリです．MIDIMANCTRLのmc_intervalに0を指定して，ユーザーが任意の割り込みにより独自の時間間隔を作り出すケースで，割り込みエントリに続いて呼び出します．

このエントリは，割り出し処理のためPICの低優先の割り込みに引き継がれます．したがって，処理中に，続く割り込み(高優先)を受け付けるので，割り込み処理を経由して2重に呼び出される可能性があります．

それを防ぐには，アプリケーション側で以前の呼び出し後の処理が完了しているかどうかを確認し，そうでなければバイパスするようプログラミングしなければなりません．

また，割り込みを受け付ける状態にあることから，スタックは十分に(1KB以上)確保する必要があります．

該当C関数　なし

MIDIマネージャ	C0H
ユーザーコールバックルーチンの登録	機能コード07H

エントリ　AH　=07H
DS：EBX　=FUNCCTRL構造体へのポインタ

リターン　EAX　=0(正常終了)
－1(すでに4個登録済)
EDX　=コールバックルーチンハンドル

説明　FUNCCTRL構造体でコールバックルーチンを登録するオペレーションです．パラメータは，次の形式で定義します．

実行後，EDXには対応するコールバックルーチンハンドルが与えられます．その値は，後述のユーザーコールバックルーチンの取得や解除の呼び出しに利用します．

(DS : EBX)

0	struct	FUNCCTRL

該当C関数　int TMM_setUserFunc(FUNCCRL *func_ctrl, int handle);

MIDI マネージャ　COH

ユーザーコールバックルーチンの取得　機能コード 08H

エントリ　AH　=08H
EDX　=コールバックルーチンハンドル
DS : EBX　=FUNCCTRL 構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　指定されたコールバックルーチンハンドル対応の，登録済みコールバックルーチンのエントリを，FUNCCTRL 構造体に読み出すオペレーションです．結果は次の形式で渡されます．

(DS : EBX)

0	struct	FUNCCTRL

該当C関数　int TMM_getUserFunc(int handle, FUNCCRL *func_ctrl);

MIDI マネージャ　COH

ユーザーコールバックルーチンの解除　機能コード 09H

エントリ　AH　=09H
EDX　=コールバックルーチンハンドル

リターン　EAX　=0(正常終了)

説　明　指定されたコールバックルーチンハンドル対応の，登録済みコールバックルーチンを解除するオペレーションです．

該当C関数　int TMM_resetUserFunc(int handle);

MIDIマネージャ　C0H

MIDIデータ出力　機能コード0AH

エントリ	AH	＝0AH
	BL	＝出力ポート
	DS：ESI	＝イベントデータへのポインタ
リターン	EAX	＝0(正常終了)

説　明

MIDIの指定ポートに，下図のイベントを出力します．このとき，イベント発生時刻の内容は無視され，データサイズで指定されたデータ長だけ転送されます．その際，内容のチェックは行われません．

該当C関数　int TMM_outputMidiEvent(Port port, char *event);

MIDIマネージャ　C0H

演奏の開始　機能コード10H

エントリ	AH	＝10H
	DL	＝0(MIDIファイル準拠フォーマット) 1(EUPファイルフオーマット)
	DH	＝0(再生) 0以外(再生と同時に録音)
	DS：ESI	＝プレイバッファへのポインタ
	ECX	＝プレイバッファサイズ
	DS：EBX	＝演奏開始点のPLACE構造体へのポインタ

リターン	EAX	=0(正常終了)

説　明

演奏を開始するオペレーションです．単なる再生だけでなく，再生と同時に録音することもできます．DS：EBXの値と，PLACE構造体の関係は次のとおりです．

(DS：EBX)

0	struct	PLACE

該当C関数

int TMM_playStart(char format, FLAG rec_flag, char *playbuffer, int size, PLACE *place);

MIDIマネージャ	C0H
演奏の終了	**機能コード11H**

エントリ	AH	=11H
リターン	EAX	=0(正常終了)
説　明	演奏を終了するオペレーションです．	
該当C関数	int TMM_playStop();	

MIDIマネージャ	C0H
演奏の一時中断	**機能コード12H**

エントリ	AH	=12H
リターン	EAX	=0(正常終了)
説　明	再開を前提に，演奏を一時中断するオペレーションです．	
該当C関数	int TMM_playPause();	

MIDIマネージャ	COH
演奏の再開	機能コード13H

エントリ　AH　=13H

リターン　EAX　=0(正常終了)

説　明　一時中断していた演奏を再開するオペレーションです.

該当C関数　int TMM_playRestart();

MIDIマネージャ	COH
演奏モードの設定	機能コード14H

エントリ　AH　=14H
DL　=0(ノーマル再生)
1(エンドレス再生)
2(ループ再生)

リターン　EAX　=0(正常終了)

説　明　演奏モードを設定します.

エンドレス再生は，演奏データが終わっても，最後の拍子とテンポを引き継いで演奏のカウントだけを続行します.

ループ再生は，演奏データが終わっても，再度ポインタを先頭に戻し，演奏を続行します.

該当C関数　int TMM_playMode(char mode);

MIDIマネージャ	COH
演奏位置の取得	機能コード16H

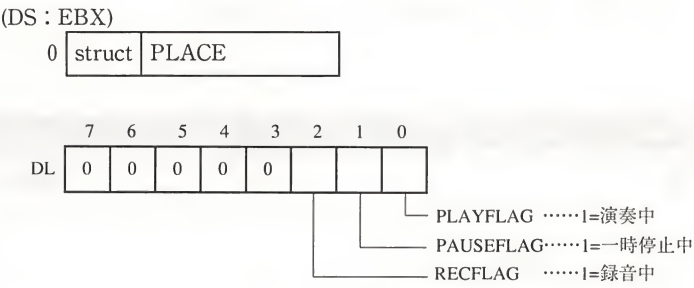
エントリ　AH　　　=16H
　　　　　DS：EDI　=演奏位置を格納するPLACE構造体へのポインタ

リターン　EAX　　=0(正常終了)
　　　　　DL　　　=演奏状態フラグ
　　　　　EBX　　=演奏ポインタ

説　明　演奏状態，演奏位置，演奏ポインタを参照して，レジスタや構造体に転送するオペレーションです．

DS：EBXとPLACE構造体の関係，および演奏状態フラグのビット構成は次のとおりです．

(DS：EBX)

0	struct	PLACE

該当C関数　int TMM_getPlayPlace(FLAG, unsigned char **ptr, PLACE *);

MIDI マネージャ	C0H
テンポの設定	機能コード 17H

エントリ　AH　＝17H
DL　＝0(1/10 BPM 単位)
　1(μs 単位)
EBX　＝テンポまたは四分音譜の長さ

リターン　EAX　＝0(正常終了)

説　明　テンポの設定を行うオペレーションで，テンポ(1分当たり拍数)または四分音譜の音長を規定します．単位は，テンポ(1/10 BPM 単位)を実際のbpmの1/10，四分音譜の音長(μs 単位)をマイクロ秒の時間で設定します．
いずれも，転送レートに換算して，5～500bpmの範囲でなければなりません．

該当C関数　int TMM_setTempo(char type, int tempo);

MIDI マネージャ	C0H
テンポの取得	機能コード 18H

エントリ　AH　＝18H
DL　＝0(1/10 BPM 単位)
　1(μs 単位)

リターン　EAX　＝0(正常終了)
EBX　＝転送レートまたは四分音譜の長さ

説　明　現在設定されているテンポまたは，四分音譜の音長を参照します．単位は，1/10 BPM 単位を実際のbpmの1/10，μs 単位をマイクロ秒の時間で表現します．

該当C関数　int TMM_getTempo(char type, int ＊tempo);

MIDIマネージャ	COH
相対テンポの設定	**機能コード 19H**

エントリ　AH　＝19H
EBX　＝相対テンポ(0～1000%)

リターン　EAX　＝0(正常終了)

説　明　現在設定されているテンポを，指定されたパーセンテージで変化させます．0%を指定するとステップ演奏モードとなり，「ステップモードの進行(機能コード 1DH)」と合わせて，任意のタイミングで進行させることができます．

該当C関数　int TMM_setRelativeTempo(int relative_tempo);

MIDIマネージャ	COH
相対テンポの取得	**機能コード 1AH**

エントリ　AH　＝1AH

リターン　EAX　＝0(正常終了)
EBX　＝相対テンポ(0～1000%)

説　明　現在設定されている相対テンポをEBXに読み取ります．

該当C関数　int TMM_setRelativeTempo(int *relative_tempo);

MIDIマネージャ	COH
EUPデータ相対テンポの設定	**機能コード 1BH**

エントリ　AH　＝1BH
EBX　＝EUP相対テンポ(－280～＋280)

リターン　EAX　＝0(正常終了)

説　明　現在設定されているEUPファイル演奏テンポを，指定されたピッチで変化させます．EBXの値は加算値です．

該当C関数　int TMM_setEupTempoOffset(int tempo_offset);

MIDI マネージャ　COH

EUP データ相対テンポの取得　機能コード 1CH

エントリ　AH　=1CH

リターン　EAX　=0(正常終了)
EBX　=EUP 相対テンポ

説　明　現在設定されている EUP 相対テンポを EBX に読み取ります.

該当C関数　int TMM_getEupTempoOffset(int *tempo _offset);

MIDI マネージャ　COH

ステップモードの進行　機能コード 1DH

エントリ　AH　=1DH
EDX　=進行クロック数

リターン　EAX　=0(正常終了)

説　明　ステップ演奏モード (「相対テンポの設定 (機能コード 19H)」で 0%を設定の場合) で，進行クロック数で指定されたクロック数だけシーケンスが進行します.
この機能を使って，ステップ録音などができます.

該当C関数　int TMM_stepModeEntry(int step_clock);

MIDIマネージャ	COH
同期モードの設定	**機能コード20H**

エントリ　AH　＝20H
DL　＝0(内蔵タイマ同期モード)
　1(MIDIクロック同期モード)
　　BL＝同期入力ポート(DL＝1のとき)
　2(SMPTE同期モード)
　　DH＝0(S-MPUタイマ)
　　　1(S-MPU外部機器)
　　　2(MIDIタイムコード)
　　BL＝同期入力ポート(DL＝2，DH＝2のとき)
　　CL＝SMPTEタイプ
　　CH＝エラー補正レベル(推奨値7)

リターン　EAX　＝0(正常終了)

説　明　同期モードを設定するオペレーションです．DLで大まかな指定を行い，さらに細かな指定を，字下げして記述しているレジスタで定義します．

内蔵タイマ同期は，FMTOWNS本体のタイマで同期をとります．MIDIクロック同期は，指定された同期入力ポートからのクロックで同期をとります．また，SMPTE同期は，Roland社Super−MPU(“S−MPU”と略す)のSMPTE READ/WRITE機能で提供されるクロックに同期し，業務用VTRなどとの同期演奏ができます．

該当C関数　int TMM_setSyncIn(char mode, PORT port, char master, int type,int level);

MIDIマネージャ	COH
SMPTE開始位置の設定	**機能コード21H**

エントリ　AH　＝21H
DS：EDI　＝スタート時刻を示すREALTIME構造体へのポインタ

リターン　EAX　＝0(正常終了)

説　明　REALTIME 構造体を使って，SMPTE 同期をする場合のスタート時刻を設定します.

DS : EDI と REALTIME 構造体の関係は次のとおりです.

(DS : EDI)

0	struct	REALTIME

該当 C 関数　int TMM_setSmpteOffset(REALTIME ＊time);

MIDI マネージャ　C0H
SMPTE 同期精度の設定　機能コード 22H

エントリ

AH　=22H
EDX　=タイムラグ (1/480μs 単位)
ECX　=補正待ち時間 (1/480μs 単位)

リターン

EAX　=0(正常終了)

説　明　SMPTE 同期をする場合の補正パラメータを変更します. このオペレーションで書き換えしないときのタイムラグの初期値は 960000(2ms), 補正待ち時間は 96000000(200ms) です.

同期の開始に当たっては, SMPTE 時間と演奏開始時刻の差を常時監視し, タイムラグの範囲に入ると演奏を開始します. タイムラグの値が小さいと, SMPTE の転送タイミングによっては同期を開始できないこともあるので注意が必要です.

演奏実時間と SMPTE 実時間との差が指定されたタイムラグの範囲内なら同期中とみなし, それを超えるずれが生じたとき補正待ち時間だけ演奏を続けます. それでもタイムラグの範囲に収まらないときは, 同期が外れたものとして扱います. このような処理は, S-MPU からの SMPTE 時間が正しく読み取れなかったり, MIDI 入出力優先処理を行った場合などでも, 同期を確保できるようにするために行われます.

該当 C 関数　int TMM_setSmpteLag(int lag, int correction);

MIDIマネージャ	C0H
S-MPU内部時間の設定	機能コード23H

エントリ　AH　=23H
DS:ESI　=REALTIME構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　REALTIME構造体を使って，S-MPUの内部時計の値を設定します．
DS:ESIとREALTIME構造体の関係は次のとおりです．

(DS:ESI)

0	struct	REALTIME

該当C関数　int TMM_setSmpuTime(REALTIME *TIME);

MIDIマネージャ	C0H
実時間からSMPTE時間への変換	機能コード24H

エントリ　AH　=24H
DS:ESI　=REALTIME構造体へのポインタ
DS:EDI　=SMPTE構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　REALTIME構造体の実時間情報を，SMPTE構造体のタイムコードに変換します．SMPTE構造体にはフレームタイプがあるので，事前にどのタイプを選定するかの情報を書き込んでおく必要があります．

DS:ESIとREALTIME構造体の関係，およびDS:EDIとSMPTE構造体の関係は次のとおりです．

(DS:ESI)

0	struct	REALTIME

(DS:EDI)

0	struct	SMPTE

該当C関数　int TMM_realtimeToSmpte(SMPTE *smpte, REALTIME *TIME);

MIDIマネージャ　C0H
SMPTE時間から実時間への変換　機能コード25H

エントリ　AH　=25H
DS : ESI　=REALTIME構造体へのポインタ
DS : EDI　=SMPTE構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　SMPTE構造体のタイムコードを，REALTIME構造体の実時間情報に変換します．

DS : EDIとSMPTE構造体の関係，およびDS : ESIとREALTIME構造体の関係は次のとおりです．

(DS : ESI)

0	struct	REALTIME

(DS : EDI)

0	struct	SMPTE

該当C関数　int TMM_smpteToRealtime(REALTIME *TIME, SMPTE *smpte);

MIDIマネージャ　C0H
リモートモードの設定　機能コード26H

エントリ　AH　=26H
DL　=0(リモートモードOFF)
　0以外(リモートモードON)

リターン　EAX　=0(正常終了)

説　明　リモートモードの設定/解除を行うオペレーションです．

リモートモードでは，同期入力から受け取ったスタート($FA)，コンティニュ($FB)，ストップ($FC)に応答します．

該当C関数　int TMM_setRemoteMode(char mode);

MIDI マネージャ　　COH

同期信号出力の設定　　機能コード27H

エントリ　AH　＝27H
DL　＝0(同期信号出力 OFF)
　　1(同期信号出力 ON)
BL　＝同期信号出力ポート

リターン　EAX　＝0(正常終了)

説　明　MIDI クロック (同期信号) を出力するスイッチオペレーションです. SMPTE 同期についてはサポートされていません.

該当C関数　int TMM_setSyncOut(char mode, PORT port);

MIDI マネージャ　　COH

メトロノームの設定　　機能コード28H

エントリ　AH　＝28H
DS：ESI　＝METRONOME 構造体へのポインタ

リターン　EAX　＝0(正常終了)

説　明　METRONOME 構造体の内容に従って, メトロノームを設定します.
DS：ESI と METRONOME 構造体の関係は次のとおりです.

(DS：ESI)

0	struct	METRONOME

該当C関数　int TMM_setMetronome(METRONOME *metronome);

MIDI マネージャ	C0H
アサインマップの設定	**機能コード 30H**

エントリ	AH	＝30H
	BL	＝入力ポート番号
	DL	＝アサインマップ番号 (0～3)
リターン	EAX	＝0(正常終了)

説　明　　指定された入力ポートについて，出力ポートへのアサインマップを設定します．

該当C関数　　int TMM_setAssignMap(PORT port, char map_number);

MIDI マネージャ	C0H
アサインマップの取得	**機能コード 31H**

エントリ	AH	＝31H
	BL	＝入力ポート番号
リターン	EAX	＝0(正常終了)
	DL	＝アサインマップ番号 (0～3)

説　明　　指定された入力ポートについて，出力ポートへのアサインマップの設定値を参照します．

該当C関数　　int TMM_getAssignMap(PORT port, char *map_number);

MIDI マネージャ　COH

アサインフィルタの設定　機能コード 32H

エントリ　AH　=32H
DL　=アサインマップ番号 (0〜3)
DS : ESI　=ASSIGNFILTER 構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　指定されたアサインマップ番号に対応するアサインフィルタを登録します.
DS : ESI と ASSIGNFILTER 構造体の関係は次のとおりです.

(DS : ESI)

0	struct	ASSIGNFILTER

該当C関数　int TMM_setAssignFilter(char map_number, ASSIGNFILTER *filter);

MIDI マネージャ　COH

アサインフィルタの取得　機能コード 33H

エントリ　AH　=33H
DL　=アサインマップ番号 (0〜3)
DS : ESI　=ASSIGNFILTER 構造体へのポインタ

リターン　EAX　=0(正常終了)

説　明　指定されたアサインマップ番号に対応するアサインフィルタの登録内容を参照します.
DS : ESI と ASSIGNFILTER 構造体の関係は次のとおりです.

(DS : ESI)

0	struct	ASSIGNFILTER

該当C関数　int TMM_getAssignFilter(char map_number, ASSIGNFILTER *filter);

MIDIマネージャ COH
出力ポートマップの設定 機能コード34H

エントリ	AH	=34H
	BL	=論理出力ポート
	DL	=物理出力ポート
リターン	EAX	=0(正常終了)

説　明　指定された論理出力ポートに対して，物理出力ポートを設定します.

該当C関数　int TMM_setOutputPortMap(PORT logical_port, PORT physical_port);

MIDIマネージャ COH
出力ポートマップの取得 機能コード35H

エントリ	AH	=35H
	BL	=論理出力ポート
リターン	EAX	=0(正常終了)
	DL	=物理出力ポート

説　明　指定された論理出力ポートに対して，設定されている物理出力ポートの値を参照します.

該当C関数　int TMM_getOutputPortMap(PORT logical_port, PORT *physical_port);

MIDIマネージャ COH
入力ポートマップの設定 機能コード36H

エントリ	AH	=36H
	DL	=物理入力ポート
	BL	=論理入力ポート
リターン	EAX	=0(正常終了)

説　明　指定された物理入力ポートに対して，論理入力ポートを設定します.

該当C関数　int TMM_setInputPortMap(PORT logical_port, PORT physical_port);

MIDIマネージャ	C0H
入力ポートマップの取得	機能コード37H

エントリ　AH　=37H
　　　　　DL　=物理入力ポート

リターン　EAX　=0(正常終了)
　　　　　BL　=論理入力ポート

説　明　指定された物理入力ポートに対して，設定されている論理入力ポートの値を参照します．

該当C関数　int TMM_getInputPortMap(PORT logicalu _port, PORT *physical_port);

MIDIマネージャ	C0H
内蔵音源の初期化	機能コード40H

エントリ　AH　=40H

リターン　EAX　=0(正常終了)

説　明　内蔵音源について，MIDIエミュレーションのための初期化をします．

該当C関数　int TMM_initInternalVoice();

MIDIマネージャ	C0H
内蔵音源のMIDIデータ出力	機能コード41H

エントリ　AH　=41H
　　　　　DL　=MIDIデータ

リターン　EAX　=0(正常終了)

説　明　内蔵音源にMIDIデータを転送します．

該当C関数　int TMM_sendInternalVoice(unsigned char data);

MIDIマネージャ	COH
内蔵音源のMIDIチャネルの設定	機能コード42H

エントリ　AH　=42H
BL　=内蔵音源チャネル
DL　=MIDIチャネル

リターン　EAX　=0(正常終了)
－3(パラメータの値が異常)

説　明　指定された内蔵音源にMIDIチャネルを割り当てます.

該当C関数　int TMM_setInternalChannel(unsigned char voice_ch, MIDICH midi_ch);

MIDIマネージャ	COH
内蔵音源のMIDIチャネルの取得	機能コード43H

エントリ　AH　=43H
BL　=内蔵音源チャネル

リターン　EAX　=0(正常終了)
－3(パラメータの値が異常)
DL　=MIDIチャネル

説　明　指定された内蔵音源に割り当てられたMIDIチャネル値を参照します.

該当C関数　int TMM_getInternalChannel(unsigned char voice_ch, MIDICH *midi_ch);

MIDIマネージャ	COH
内蔵音源のマスタボリュームの設定	機能コード44H

エントリ　AH　=44H
DL　=マスタボリューム(1～127)

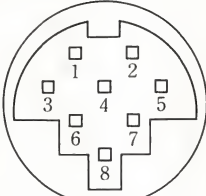
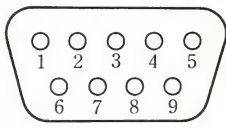
リターン　EAX　=0(正常終了)
－3(パラメータの値が異常)

説　明　指定された内蔵音源のマスタボリュームを設定します.

該当C関数　int TMM_setInternalVolume(char volume);

MIDIマネージャ	C0H
内蔵音源のマスタボリュームの取得	機能コード45H

エントリ	AH	=45H
リターン	EAX	=0(正常終了)
		−3(パラメータの値が異常)
	DL	=マスタボリューム(1〜127)
説　明		指定された内蔵音源のマスタボリューム設定値を参照します.
該当C関数		int TMM_getInternalVolume(char volume);

付　録

付録A

各種コネクタの仕様とピン配置

A.1　キーボードコネクタ

端子番号	信号名(内容)	ピンコネクション
1	DG(デジタルグランド)	
2	DG(デジタルグランド)	
3	KSD(キーボードから送られてくるシリアルデータ)	
4	CTLR(キーボードへ送るコントロールデータ)	
5	OPEN(未接続)	
6	OPEN(未接続)	
7	VCC(電源)	
8	VCC(電源)	

A.2　パッド&マウスコネクタ

端子番号	信号名(内容)	ピンコネクション
1	FWD(上方向キー)	
2	BACK(下方向キー)	
3	LEFT(左方向キー)	
4	RIGHT(右方向キー)	
5	VCC(電源)	
6	TRIG1(トリガボタン1)	
7	TRIG2(トリガボタン2)	
8	COM(帰還端子)	
9	GND(グランド)	

A.3 RS-232C コネクタ

RDBD-25S-LNA(05) ヒロセ電機製または同等品(通称 D-SUB 25P)

端子番号	信 号 名	信号方向	機 能	ピンコネクション
1	FG	——	フレームグランド	
2	SD	出力	シリアル送信データ	
3	RD	入力	シリアル受信データ	
4	RS	出力	送信要求信号	
5	CS	入力	送信許可信号	
6	DR	入力	DCE 電源投入示唆	
7	SG	——	シグナルグランド	
8	CD	入力	キャリア確定示唆	
9	(NC)	——	開放	
10	(NC)	——	開放	
11	ONOF2	入力	本装置電源制御信号	
12	(NC)	——	開放	
13	(NC)	——	開放	
14	(NC)	——	開放	
15	ST2	入力	送信タイミング信号	
16	(NC)	——	開放	
17	RT	入力	受信タイミング信号	
18	(NC)	——	開放	
19	(NC)	——	開放	
20	ER	出力	本装置電源投入示唆	
21	(NC)	——	開放	
22	CI	入力	呼び出し信号	
23	(NC)	——	開放	
24	ST1	出力	送信タイミング信号	
25	(NC)	——	開放	

A.4 プリンタコネクタ

57LE-40240-7700(D12)　第一電子製または同等品(通称　アンフェノール型24P)

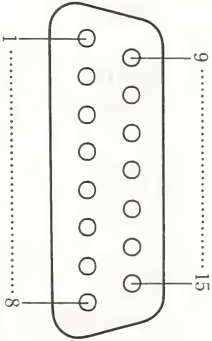
端子番号	信号名	信号方向	機能	ピンコネクション
1	＊PSTB	出力	ストローブ信号	
2	PD1	出力	プリントデータ	
3	PD2	出力	〃	
4	PD3	出力	〃	
5	PD4	出力	〃	
6	PD5	出力	〃	
7	PD6	出力	〃	
8	PD7	出力	〃	
9	PD8	出力	〃	
10	＊ACK	入力	応答信号(READY)	
11	BUSY	入力	〃　(BUSY)	
12	PE	入力	用紙なし検出信号	
13	SLCT	入力	セレクト信号	
14	RINF1	入力	プリンタ状態信号	
15	RINF2	入力	〃	
16	RINF3	入力	〃	
17	+5V	入力	電源信号	
18	＊INPRM	出力	リセット信号	
19	＊EXPRM	出力	リセット信号	
20	＊FAULT	入力	エラー通知信号	
21	＊FUSE	入力	ヒューズ断検出信号	
22	THSN	入力	熱アラーム検出信号	
23	GND	——	グランド	
24	GND	——	グランド	

＊は負論理を示す.

A.5 フロッピィコネクタ

PCS-E50LMD　本多通信工業製または同等品(通称　PCS 50P)

端子番号	信 号 名	信号方向	機　　能	ピンコネクション
1	* HSEL	出力	ヘッド選択信号	
2	* INUSE	出力	ドアロック/ランプ制御信号	
3	* RD	入力	リードデータ信号	
4	* RDY2	入力	ドライブ2からのレディ信号	
5	* DSKSNS	入力	両面媒体検出信号	
6	* INDEX	入力	インデックスホール検出信号	
7	* WD	出力	ライトデータ信号	
8	* DRVSNS	入力	ドライブ識別信号	
9	* WG	出力	ライトゲート信号	
10	* FUN	入力	ファイルアンセーフ検出信号	
11	* WP	入力	ライトプロテクト信号	
12	* TRO	入力	トラック検出信号	
13	* USRST	出力	ファイルアンセーフリセット信号	
14	* LCR	出力	書き込み電流切換信号	
15	* STEP	出力	ヘッド移動信号	
16	* DIR	出力	ヘッド移動方向指定信号	
17	* HLD	出力	ヘッドロード信号	
18	* DS2	出力	ドライブ2の選択信号	
19	* DS3	出力	ドライブ3の選択信号	
20	* DS2	出力	ドライブ2の選択信号	
21	* DS3	出力	ドライブ3の選択信号	
22	* SFLT	出力	リードフィルタ切り換え信号	
23	* RDY3	入力	ドライブ3からのレディ信号	
24	* RDY2	入力	ドライブ2からのレディ信号	
25	* RDY3	入力	ドライブ3からのレディ信号	
26	* LOWSPD	出力	回転数切り換え信号	
27〜49	GND	――――	グランド	
50	* MOTORON	出力	モータのON/OFF信号	

*は負論理を示す.

A.6 アナログRGBコネクタ

FMTOWNS 本体装備のアナログ RGB コネクタ

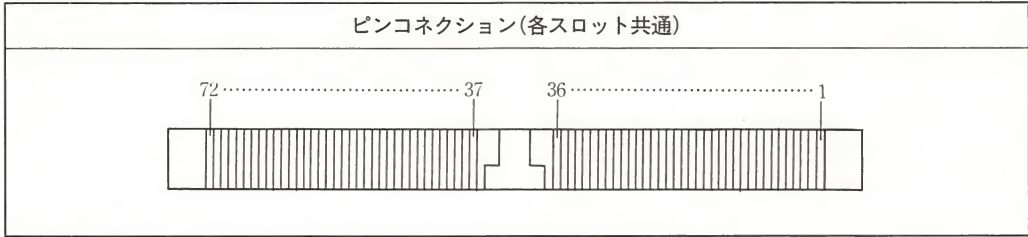
端子番号	信号名	説明	ピンコネクション
1	VROUT	アナログ RGB 出力 赤　0.7Vp-p　75Ω 終端	
2	VGND	GND	
3	VGOUT	アナログ RGB 出力 緑　0.7Vp-p　75Ω 終端	
4	VGND	GND	
5	VBOUT	アナログ RGB 出力 青　0.7Vp-p　75Ω 終端	
6	VGND	GND	
7	CSYNCOUT	複合同期信号出力　1Vp-p　75Ω 終端	
8	VGND	GND	
9	Ys	パソコン出力/外部入力出力の区別	
10	AVCONT	AV コントロール　TTL レベル	
11	———		
12	VGND	GND	
13	SCLK	ドットロック信号の原発振	
14	* HSYNC	水平同期信号出力　TTL レベル	
15	* VSYNC	垂直同期信号出力　TTL レベル	

*は負論理を示す.

ビデオカードに装備のアナログ RGB コネクタ

端子番号	信 号 名	説　　明	ピンコネクション
1	VROUT	スーパーインポーズアナログ RGB 出力 赤　0.7Vp-p　75Ω 終端	
2	VGND	GND	
3	VGOUT	スーパーインポーズアナログ RGB 出力 緑　0.7Vp-p　75Ω 終端	
4	VGND	GND	
5	VBOUT	スーパーインポーズアナログ RGB 出力 青　0.7Vp-p　75Ω 終端	
6	VGND	GND	
7	CSYNCOUT	複合同期信号出力　1Vp-p　75Ω 終端	
8	VGND	GND	
9	———		
10	AVCONT	AV コントロール　TTL レベル	
11	———		
12	VGND	GND	
13	———		
14	* HSYNC	水平同期信号出力　TTL レベル	
15	* VSYNC	垂直同期信号出力　TTL レベル	

* は負論理を示す.

A.7　拡張 RAM モジュールコネクタ

スロット1

端子番号	信号名	入出力	信号レベル	信 号 内 容
1	GND	——	GND	グランド
2	D0	IN/OUT	TTL	データ信号
3	D16	IN/OUT	TTL	データ信号
4	D1	IN/OUT	TTL	データ信号
5	D17	IN/OUT	TTL	データ信号
6	D2	IN/OUT	TTL	データ信号
7	D18	IN/OUT	TTL	データ信号
8	D3	IN/OUT	TTL	データ信号
9	D19	IN/OUT	TTL	データ信号
10	+5V	——	+5V	電源 +5V
11	(NC)	——	——	——
12	MA0il	OUT	TTL	アドレス信号
13	MA1il	OUT	TTL	アドレス信号
14	MA2il	OUT	TTL	アドレス信号
15	MA3il	OUT	TTL	アドレス信号
16	MA4il	OUT	TTL	アドレス信号
17	MA5il	OUT	TTL	アドレス信号
18	MA6il	OUT	TTL	アドレス信号
19	(NC)	——	——	——
20	D4	IN/OUT	TTL	データ信号
21	D20	IN/OUT	TTL	データ信号
22	D5	IN/OUT	TTL	データ信号
23	D21	IN/OUT	TTL	データ信号
24	D6	IN/OUT	TTL	データ信号
25	D22	IN/OUT	TTL	データ信号
26	D7	IN/OUT	TTL	データ信号
27	D23	IN/OUT	TTL	データ信号
28	MA7il	OUT	TTL	アドレス信号
29	(NC)	——	——	——
30	+5V	——	+5V	電源 +5V
31	MA8il	OUT	TTL	アドレス信号
32	(NC)	——	——	——
33	＊RAS3	——	+5V	ロウアドレスストローブ(未使用)
34	＊RAS1-2M	OUT	TTL	ロウアドレスストローブ1-2M用
35	MP2	——	——	パリティ(未使用)
36	MP0	——	——	パリティ(未使用)
37	MP1	——	——	パリティ(未使用)
38	MP3	——	——	パリティ(未使用)
39	GND	——	GND	グランド

端子番号	信号名	入出力	信号レベル	信　号　内　容
40	＊CAS0	OUT	TTL	コラムアドレスストローブ 0
41	＊CAS2	OUT	TTL	コラムアドレスストローブ 2
42	＊CAS3	OUT	TTL	コラムアドレスストローブ 3
43	＊CAS1	OUT	TTL	コラムアドレスストローブ 1
44	＊RAS1-2M	OUT	TTL	ロウアドレスストローブ　1-2M 用
45	＊RAS1	——	+5V	ロウアドレスストローブ(未使用)
46	(NC)	——	——	——
47	＊MWTCB1	OUT	TTL	メモリライトコマンド
48	(NC)	——	——	——
49	D8	IN/OUT	TTL	データ信号
50	D24	IN/OUT	TTL	データ信号
51	D9	IN/OUT	TTL	データ信号
52	D25	IN/OUT	TTL	データ信号
53	D10	IN/OUT	TTL	データ信号
54	D26	IN/OUT	TTL	データ信号
55	D11	IN/OUT	TTL	データ信号
56	D27	IN/OUT	TTL	データ信号
57	D12	IN/OUT	TTL	データ信号
58	D28	IN/OUT	TTL	データ信号
59	+5V	——	+5V	電源　+5V
60	D29	IN/OUT	TTL	データ信号
61	D13	IN/OUT	TTL	データ信号
62	D30	IN/OUT	TTL	データ信号
63	D14	IN/OUT	TTL	データ信号
64	D31	IN/OUT	TTL	データ信号
65	D15	IN/OUT	TTL	データ信号
66	(NC)	——	——	——
67	ID0	——	——	メモリモジュールの種類を示す(未使用)
68	ID1	——	——	メモリモジュールの種類を示す(未使用)
69	(NC)	——	——	——
70	GND	——	GND	グランド
71	(NC)	——	——	——
72	GND	——	GND	グランド

＊は負論理を示す.

スロット2

端子番号	信号名	入出力	信号レベル	信　号　内
1	GND	——	GND	グランド
2	D0	IN/OUT	TTL	データ信号
3	D16	IN/OUT	TTL	データ信号
4	D1	IN/OUT	TTL	データ信号
5	D17	IN/OUT	TTL	データ信号
6	D2	IN/OUT	TTL	データ信号
7	D18	IN/OUT	TTL	データ信号
8	D3	IN/OUT	TTL	データ信号
9	D19	IN/OUT	TTL	データ信号
10	+5V	——	+5V	電源 +5V
11	(NC)	——	——	——
12	MA0i	OUT	TTL	アドレス信号
13	MA1i	OUT	TTL	アドレス信号
14	MA2i	OUT	TTL	アドレス信号
15	MA3i	OUT	TTL	アドレス信号
16	MA4i	OUT	TTL	アドレス信号
17	MA5i	OUT	TTL	アドレス信号
18	MA6i	OUT	TTL	アドレス信号
19	(NC)	——	——	——
20	D4	IN/OUT	TTL	データ信号
21	D20	IN/OUT	TTL	データ信号
22	D5	IN/OUT	TTL	データ信号
23	D21	IN/OUT	TTL	データ信号
24	D6	IN/OUT	TTL	データ信号
25	D22	IN/OUT	TTL	データ信号
26	D7	IN/OUT	TTL	データ信号
27	D23	IN/OUT	TTL	データ信号
28	MA7i	OUT	TTL	アドレス信号
29	(NC)	——	——	——
30	+5V	——	+5V	電源 +5V
31	MA8i	OUT	TTL	アドレス信号
32	(NC)	——	——	——
33	* RAS3-4M	OUT	TTL	ロウアドレスストローブ 3-4MB 用
34	* RAS2-3M	OUT	TTL	ロウアドレスストローブ 2-3MB 用
35	MP2	——	——	パリティ(未使用)
36	MP0	——	——	パリティ(未使用)
37	MP1	——	——	パリティ(未使用)
38	MP3	——	——	パリティ(未使用)
39	GND	——	GND	グランド

端子番号	信号名	入出力	信号レベル	信 号 内 容
40	＊CAS0	OUT	TTL	コラムアドレスストローブ0
41	＊CAS2	OUT	TTL	コラムアドレスストローブ2
42	＊CAS3	OUT	TTL	コラムアドレスストローブ3
43	＊CAS1	OUT	TTL	コラムアドレスストローブ1
44	＊RAS2-3M	OUT	TTL	ロウアドレスストローブ　2-3MB用
45	＊RAS3-4M	OUT	TTL	ロウアドレスストローブ　3-4MB用
46	(NC)	——	——	——
47	＊MWTCB1	OUT	TTL	メモリライトコマンド
48	(NC)	——	——	——
49	D8	IN/OUT	TTL	データ信号
50	D24	IN/OUT	TTL	データ信号
51	D9	IN/OUT	TTL	データ信号
52	D25	IN/OUT	TTL	データ信号
53	D10	IN/OUT	TTL	データ信号
54	D26	IN/OUT	TTL	データ信号
55	D11	IN/OUT	TTL	データ信号
56	D27	IN/OUT	TTL	データ信号
57	D12	IN/OUT	TTL	データ信号
58	D28	IN/OUT	TTL	データ信号
59	+5V	——	+5V	電源　+5V
60	D29	IN/OUT	TTL	データ信号
61	D13	IN/OUT	TTL	データ信号
62	D30	IN/OUT	TTL	データ信号
63	D14	IN/OUT	TTL	データ信号
64	D31	IN/OUT	TTL	データ信号
65	D15	IN/OUT	TTL	データ信号
66	(NC)	——	——	——
67	ID0	——	——	メモリモジュールの種類を示す(未使用)
68	ID1	——	——	メモリモジュールの種類を示す(未使用)
69	(NC)	——	——	——
70	GND	——	GND	グランド
71	(NC)	——	——	——
72	GND	——	GND	グランド

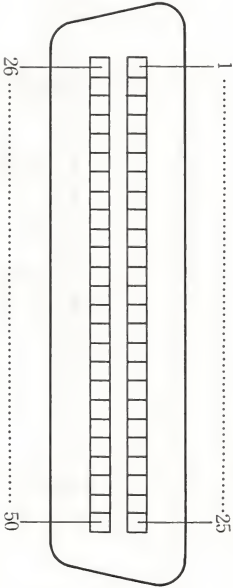
＊は負論理を示す.

スロット3

端子番号	信号名	入出力	信号レベル	信　号　内
1	GND	——	GND	グランド
2	D0	IN/OUT	TTL	データ信号
3	D16	IN/OUT	TTL	データ信号
4	D1	IN/OUT	TTL	データ信号
5	D17	IN/OUT	TTL	データ信号
6	D2	IN/OUT	TTL	データ信号
7	D18	IN/OUT	TTL	データ信号
8	D3	IN/OUT	TTL	データ信号
9	D19	IN/OUT	TTL	データ信号
10	+5V	——	+5V	電源　+5V
11	(NC)	——	——	——
12	MA0i	OUT	TTL	アドレス信号
13	MA1i	OUT	TTL	アドレス信号
14	MA2i	OUT	TTL	アドレス信号
15	MA3i	OUT	TTL	アドレス信号
16	MA4i	OUT	TTL	アドレス信号
17	MA5i	OUT	TTL	アドレス信号
18	MA6i	OUT	TTL	アドレス信号
19	(NC)	——	——	——
20	D4	IN/OUT	TTL	データ信号
21	D20	IN/OUT	TTL	データ信号
22	D5	IN/OUT	TTL	データ信号
23	D21	IN/OUT	TTL	データ信号
24	D6	IN/OUT	TTL	データ信号
25	D22	IN/OUT	TTL	データ信号
26	D7	IN/OUT	TTL	データ信号
27	D23	IN/OUT	TTL	データ信号
28	MA7i	OUT	TTL	アドレス信号
29	(NC)	——	——	——
30	+5V	——	+5V	電源　+5V
31	MA8i	OUT	TTL	アドレス信号
32	(NC)	——	——	——
33	* RAS5-6M	OUT	TTL	ロウアドレスストローブ　5-6MB 用
34	* RAS4-5M	OUT	TTL	ロウアドレスストローブ　4-5MB 用
35	MP2	——	——	パリティ(未使用)
36	MP0	——	——	パリティ(未使用)
37	MP1	——	——	パリティ(未使用)
38	MP3	——	——	パリティ(未使用)
39	GND	——	GND	グランド

端子番号	信号名	入出力	信号レベル	信 号 内
40	* CAS0	OUT	TTL	コラムアドレスストローブ0
41	* CAS2	OUT	TTL	コラムアドレスストローブ2
42	* CAS3	OUT	TTL	コラムアドレスストローブ3
43	* CAS1	OUT	TTL	コラムアドレスストローブ1
44	* RAS4-5M	OUT	TTL	ロウアドレスストローブ 4-5M用
45	* RAS5-6M	OUT	TTL	ロウアドレスストローブ 5-6M用
46	(NC)	——	——	——
47	* MWTCB1	OUT	TTL	メモリライトコマンド
48	(NC)	——	——	——
49	D8	IN/OUT	TTL	データ信号
50	D24	IN/OUT	TTL	データ信号
51	D9	IN/OUT	TTL	データ信号
52	D25	IN/OUT	TTL	データ信号
53	D10	IN/OUT	TTL	データ信号
54	D26	IN/OUT	TTL	データ信号
55	D11	IN/OUT	TTL	データ信号
56	D27	IN/OUT	TTL	データ信号
57	D12	IN/OUT	TTL	データ信号
58	D28	IN/OUT	TTL	データ信号
59	+5V	——	+5V	電源 +5V
60	D29	IN/OUT	TTL	データ信号
61	D13	IN/OUT	TTL	データ信号
62	D30	IN/OUT	TTL	データ信号
63	D14	IN/OUT	TTL	データ信号
64	D31	IN/OUT	TTL	データ信号
65	D15	IN/OUT	TTL	データ信号
66	(NC)	——	——	——
67	ID0	——	——	メモリモジュールの種類を示す(未使用)
68	ID1	——	——	メモリモジュールの種類を示す(未使用)
69	(NC)	——	——	——
70	GND	——	GND	グランド
71	(NC)	——	——	——
72	GND	——	GND	グランド

* は負論理を示す.

A.8 ビデオカードコネクタ

ビデオカードを装着するコネクタ

端子番号	信号名	端子番号	信号名	ピンコネクション
A-1	DB0	B-1	DB1	
2	DB2	2	DB3	
3	DB4	3	DG	
4	DR0	4	DR1	
5	DR2	5	DR3	
6	DR4	6	DG	
7	DG0	7	DG1	
8	DG2	8	DG3	
9	DG4	9	DG	
10	SFTCKAD	10	DG	
11	SCLKA	11	DG	
12	YS	12	YM	
13	VCRDEN	13	SCEN	
14	FR2	14	FR3	
15	FIELD	15	SC5V	
16	VG	16	*HSYNC	
17	*VSYNC	17	VG	
18	*EHSYNC	18	*ECSYNC	
19	VG	19	CVVIDEO	
20	VG	20	EVIDEO	
21	VG	21	PCROUT	
22	VG	22	PCGOUT	
23	VG	23	PCBOUT	
24	VG	24	VG	
25	*ESYN	25	VG	
26	―――	26	VG	
27	―――	27	VG	
28	―――	28	VG	
29	―――	29	+12V	
30	D+5V	30	D+5V	
31	―――	31	―――	
32	+12V	32	−12V	

*は負論理を示す.

A.9 SCSIコネクタ

SCSI カード(オプション)に付属

端子番号	信号名	信号方向	端子番号	信号名	信号方向	ピンコネクション
1	GND	——	26	* DB0	入出力	
2	GND	——	27	* DB1	入出力	
3	GND	——	28	* DB2	入出力	
4	GND	——	29	* DB3	入出力	
5	GND	——	30	* DB4	入出力	
6	GND	——	31	* DB5	入出力	
7	GND	——	32	* DB6	入出力	
8	GND	——	33	* DB7	入出力	
9	GND	——	34	* DBP	入出力	
10	GND	——	35	GND	——	
11	GND	——	36	GND	——	
12	GND	——	37	GND	——	
13	OPEN	——	38	* TERMPWR	出力	
14	GND	——	39	GND	——	
15	GND	——	40	GND	——	
16	GND	——	41	* ATM	出力	
17	GND	——	42	GND	——	
18	GND	——	43	* BSY	入力	
19	GND	——	44	* ACK	出力	
20	GND	——	45	* RST	入力	
21	GND	——	46	* MSG	出力	
22	GND	——	47	* SEL	出力	
23	GND	——	48	* C/D	入力	
24	GND	——	49	* REQ	入力	
25	GND	——	50	* I/O	入力	

* は負論理を示す.

A.10 I/O拡張ユニットスロットコネクタ

このバスは，80386，80286を搭載した機種のための共通バスであり，以下のような特長があります．

●特長

・16ビットデータバス
・24ビットアドレスバス(アドレス空間　16MB)
・CPUとの整合性のよいバス

●信号線

このバスの信号線は，次のような6グループの信号線からなります．

データバス

メモリ，I/Oとのデータの授受のための双方向の信号線で，16ビットの幅を持ちます．

アドレスバス

メモリ，I/Oをアクセスする場合のアドレスを指定する信号で，バスマスタからメモリ，I/Oへの単方向の信号線です．アドレスは，バイト単位に割り当てられ，24本で16MBのアドレス空間を持ちます．

アクセス制御信号

メモリ，I/Oをアクセスする際の制御を行う信号線です．

割り込み信号

I/OからCPUに割り込むための信号線です．

制御信号

DMAやバスマスタの切り換え，メモリの制御，システムリセットを行う信号線です．

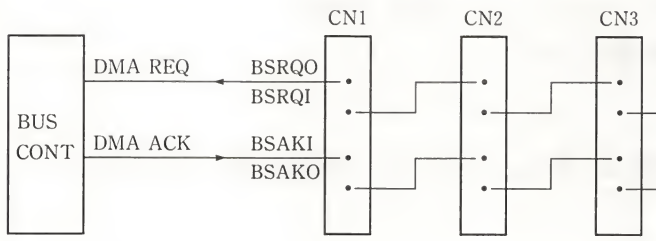
クロック信号

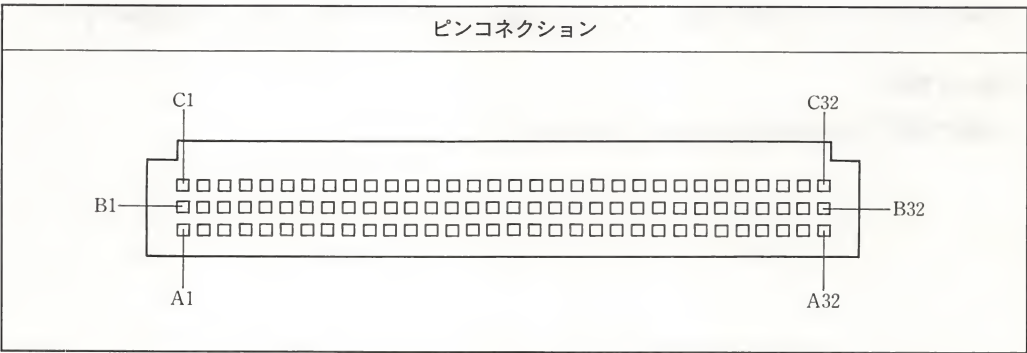
回路を動作させるためのタイミングを与えます．

●コネクタの形状とピン配列

拡張カード側のコネクタはヒロセ電機製　PCN-10A-96P-2.54DS 相当品を使用します.

DMA(バスマスタ)信号の接続

バスマスタ方式により DMA を行う場合の, 信号線の接続関係を示します. CN1 は装置前面より向かって一番左側にあり, 最も高い優先順位を持ちます. したがって, この方式で DMA を行うカードは, CN1 方向に向けた順序で実装することが必要です.

DMA を使用しないカードの信号線処理

上記の理由により, DMA を使用しない拡張カードは, 以下の接続をカード内で行い, DMA を行おうとするカードのために信号を素通りさせてください.

ピンと信号名

端子番号	信 号 名	端子番号	信 号 名	端子番号	信 号 名
A-1	+12V	B-1	+12V	C-1	−12V
2	A 0	2	予約済	2	D0
3	A 1	3	予約済	3	D1
4	A 2	4	* IOS6	4	D2
5	A 3	5	* IOS7	5	D3
6	A 4	6	0V	6	D4
7	A 5	7	* NMI	7	D5
8	A 6	8	予約済	8	D6
9	A 7	9	予約済	9	D7
10	0 V	10	* INT3	10	0V
11	A 8	11	予約済	11	D8
12	A 9	12	* INT4	12	D9
13	A10	13	* MRDC	13	D10
14	A11	14	* MWTC	14	D11
15	A12	15	0 V	15	D12
16	A13	16	* IORC	16	D13
17	A14	17	* IOWC	17	D14
18	A15	18	* WAITR	18	D15
19	0 V	19	* DRQ3	19	0 V
20	A16	20	* DACK3	20	
21	A17	21	予約済	21	* INT14
22	A18	22	予約済	22	* INT10
23	A19	23	* INTA	23	予約済
24	A20	24	0 V	24	* BSRQO
25	A21	25	* CLK	25	* BSRQI
26	A22	26	* INT5	26	* BSAKI
27	A23	27	* RFADS	27	* BSAKO
28	0 V	28	予約済	28	0 V
29	*BHE	29	予約済	29	LINECK
30	予約済	30	0 V	30	* INT8
31	+ 5 V	31	* RST	31	+ 5 V
32	+ 5 V	32	* MEMINH	32	+ 5 V

* は負論理を示す.

●信号線の機能

<table>
<tr><th>信号名</th><th>略称</th><th>方向</th><th>機 能</th></tr>
<tr><td>ADDRESSBUS</td><td>A0～A23</td><td>O/I</td><td>バスマスタ(通常はCPUまたはDMAC)からのアドレス信号.
H――1
L――0</td></tr>
<tr><td>DATABUS</td><td>D0～D15</td><td>I/O</td><td>CPU，メモリおよびI/O間のデータ転送記号.
H――1
L――0</td></tr>
<tr><td>BUS HIGH
ENABLE</td><td>* BHE</td><td>O/I</td><td>ハイバイトのアクセスが有効であることを示す．データバスのアクセス単位をアドレスバスのアクセス単位をアドレスバスのA0とともに定義する信号線で，バスマスタから出力される.
<table>
<tr><th>* BHE</th><th>A0</th><th>アクセス単位</th></tr>
<tr><td>L</td><td>L</td><td>ワード</td></tr>
<tr><td>L</td><td>H</td><td>バイト(上位8bit : D8～D15)</td></tr>
<tr><td>H</td><td>L</td><td>バイト(下位8bit : D0～D7)</td></tr>
<tr><td>H</td><td>H</td><td>禁止</td></tr>
</table></td></tr>
<tr><td>MEMORY
READ
COMMAND</td><td>* MRDC</td><td>O/I</td><td>メモリからのデータ読み出しサイクルを示す信号線で，バスマスタから出力される.
L――メモリリード</td></tr>
<tr><td>MEMORY
WRITE
COMMAND</td><td>* MWTC</td><td>O/I</td><td>メモリへのデータ書き込みサイクルを示す信号線で，バスマスタから出力される.
L――メモリライト</td></tr>
<tr><td>I/O
READ
COMMAND</td><td>* IORC</td><td>O/I</td><td>I/Oからのデータ読み出しサイクルを示す信号線で，バスマスタから出力される.
L――I/Oリード</td></tr>
<tr><td>I/O
WRITE
COMMAND</td><td>* IOWC</td><td>O/I</td><td>I/Oへのデータ書き込みサイクルを示す信号線で，バスマスタから出力される.
L――I/Oライト</td></tr>
<tr><td>WAIT
REQUEST</td><td>* WAITR</td><td>I/O</td><td>バスサイクルを延長させるためのウエイト要求信号で，スレーブからバスマスタに対して出力される．Lでウエイトサイクルが挿入される.
L⇨Hの変化は，* CLKに同期させる必要がある.</td></tr>
<tr><td>DMA
REQUEST</td><td>DRQ3</td><td>I</td><td>拡張I/OからのDMA要求.
H――DMA要求あり</td></tr>
<tr><td>DMA
ACKNOWLEGE</td><td>* DACK3</td><td>O</td><td>拡張I/Oに対するDMA許可信号.
L――DMA許可</td></tr>
<tr><td>BUS REQUE-
ST OUT</td><td>* BSRQO</td><td>I</td><td>バス使用権要求信号. * BSRQIとデイジーチェイン接続される.バスリクエスト機能を使用しない場合でも，* BSRQIと短絡しなければならない.</td></tr>
<tr><td>BUS REQUE-
ST IN</td><td>* BSRQI</td><td>O</td><td>バス使用権要求信号. * BSRQOとデイジーチェイン接続される．バスリクエスト機能を使用しない場合でも，* BSRQOと短絡しなければならない.</td></tr>
<tr><td>BUS ACKNO-
OWLEGE IN</td><td>* BSAKI</td><td>O</td><td>バス使用許可信号. * BSAKOとデイジーチェイン接続される．バスリクエスト機能を使用しない場合でも，* BSAKOと短絡しなければならない.</td></tr>
<tr><td>BUS ACKNO-
OWLEGE OUT</td><td>* BSAKO</td><td>I</td><td>バス使用許可信号. * BSAKIとデイジーチェイン接続される.バスリクエスト機能を使用しない場合でも，* BSAKIと短絡しなければならない.</td></tr>
</table>

信号名	略称	方向	機　能
INTERRUPT REQUEST	* INT3-5 * INT8 * INT10 * INT14	I	拡張I/Oからの割り込み要求信号線. Lで割り込み要求ありを示す.
INTERRUPT ACKNOWLEGE	* INTA	O	割り込み要求に対する応答信号.
MON MASKABLE INTERRUPT	* NMI	I	ノンマスカブル割り込みの要求信号. この割り込みはプログラムによるマスクはできない. H⇨Lへのエッジで割り込みが発生する.
I/O SELECT	* IOS6-7	O	I/Oアドレスデコード信号. * IOS6＝L――0C00～00FFHへのI/Oアクセス * IOS7＝L――0E00～OFFFHへのI/Oアクセス
REFRESH CLOCK	* RFADS	O	DRAMのリフレッシュクロック.
MEMORY INHIBIT	* MEMINH	I	RAMおよびROMの禁止信号. 本信号をLとすることによってROM/RAMがディセーブルされる.
CLOCK	* CLK	O	8MHzのクロック.
LINE CLOCK	LINECK	O	1.2288MHzの回線用クロック.
RESET	* RST	O	システムのイニシャルリセット信号. L――RESET

方向は, 拡張スロットからメインボードに入る信号をINPUT, メインボードから, 拡張スロットへ出る信号をOUTPUTとして表現している.

このため, BSRQI/BSRQO/BSAKI/BSAKOは信号線名称と, 方向の記述が異なるので注意が必要である.

また, アドレスバスと, コントロール信号(* MRDC, * MWTC, * IORC, * IOWC, * WAITR)は, 拡張カードがバスマスタとなった場合には, 方向が反転するため, /で分けて表現してあり, 前半がスレーブの場合, 後半がマスタの場合である.

極性は, *の付いているものは負論理の信号であり, その他は正論理だが, 表中に指定のあるものは, その指定に従う.

付録 B

サンプルプログラム

プログラム作成時の参考として，BIOS を使用したプログラムのアセンブラソースを 2 つ紹介します．

CD 演奏プログラム
描画プログラム

ここでとり上げたCD 演奏プログラムは，CD の 3 曲目を演奏するプログラムで，リアル BIOS である CD-ROM BIOS を使用しています．

描画プログラムは，円と矩形を描画するもので，ネイティブの BIOS であるグラフィック BIOS を使用しています．

どちらのプログラムも，FMTOWNS 用のアプリケーション開発キット中の 386ASM，386LINK によりアセンブル，リンクを行い，拡張子“EXP”の実行ファイルを作成します．実行ファイルは，TOWNS MENU 上でファイル名をクリックすることにより実行されます．

B.1 CD 演奏プログラム

このプログラムは，DOS-Extender の拡張ファンクションを用いてアクセスしていますので，プログラムがユーザーメモリの 1MB の範囲内にある必要があります．このままリンクする場合は問題ありませんが，サブルーチンとして使用する場合には，このモジュールがユーザーメモリの先頭の 1MB 以内に格納されるようにしなければなりません．

なお，このプログラムを単独で実行するには，4KB 程度のテンポラリスタックが必要ですので，リンク時に 386link のリンクオプションで次のように指定してください．

```
A>386link　プログラム名　-stack　4096
```

```
    .386p

TOWNS   equ 0C0h

DEVNO   equ TOWNS

TBIOS   equ 0110h
SND equ 080h

code segment dword 'CODE' use32 ; 32bit コード宣言
    assume  cs:code,ds:data
;
;   コード領域
;
    mov ax,ds
    mov es,ax

;
;   ＴＯＣ情報の読み込み
;
TOC_READ:
    mov ax,250Fh ; 386ﾈｲﾃｨﾌﾞﾓｰﾄﾞｱﾄﾞﾚｽのMS-DOSｱﾄﾞﾚｽへの変換
    lea ebx,TOCDATA  ; 変換する386ﾈｲﾃｨﾌﾞﾓｰﾄﾞｱﾄﾞﾚｽ
    mov ecx,303      ; ﾃﾞｰﾀ長
    int 21h

    jc  error

    mov di,cx
    ror ecx,16
    mov DSWD,cx

    mov ah,54h       ; ｺﾝﾊﾟｸﾄﾃﾞｨｽｸ情報の読み出し
    mov al,DEVNO ; ﾃﾞﾊﾞｲｽ番号
    mov ch,00h
    mov cl,00h       ; TOC (Table of Contents)

    mov EAXDWD,eax
    mov bx,93h
    mov RINT,bx      ; 割り込み番号

    mov ax,2511h ; ﾘｱﾙﾓｰﾄﾞ割り込みの発行
    lea edx,PARABLK  ; ﾊﾟﾗﾒｰﾀﾌﾞﾛｯｸのﾎﾟｲﾝﾀ
    int 21h

    cmp ah,00h
    je  short CD_PLAY3   ; TOC情報の読み込み正常終了
    cmp cx,80h
    je  short TOC_READ   ; ﾒﾃﾞｨｱ交換されたためﾘﾄﾗｲ
    cmp ah,10h
    jne error        ; 音楽演奏中以外のエラー

    mov ah,55h       ; 音楽演奏一時停止
    mov al,DEVNO
    mov ch,00h
    int 93h
    jmp TOC_READ

;
;   ＣＤ－ＤＡ３曲目を演奏
;

CD_PLAY3:
    mov ax,TBIOS ; ROMのセレクタをFSにセット
    mov fs,ax
    mov ah,46h       ; 電子ボリュームミュートのセット
    mov bl,00110000b ; CDのLEFTとRIGHTだけミュートを解除
    call pword ptr fs:[SND]    ; サウンドBIOSを呼ぶ
    mov ah,49h       ; 電子ボリューム全ミュートのセット
    mov dl,0ffh      ; ミュート解除
    call pword ptr fs:[SND]    ; サウンドBIOSを呼ぶ

    lea esi,TOCDATA  ; TOCﾃﾞｰﾀ先頭ｱﾄﾞﾚｽ
    lea edi,PLAY3    ; 演奏ﾊﾟﾗﾒｰﾀｱﾄﾞﾚｽ

    mov al,[esi+12]  ; ３曲目演奏開始時間(分)
    mov [edi+00],al
    mov al,[esi+13]  ; ３曲目演奏開始時間(秒)
    mov [edi+01],al
    mov al,[esi+14]  ; ３曲目演奏開始時間(FRAME)
    mov [edi+02],al

    mov al,[esi+02]  ; 最終ﾄﾗｯｸ
    cmp al,3
    jl  error        ; ３曲未満しかない
    je  short Disc_Time   ; ３曲しかない

Track_Time:
    mov dl,[esi+15]  ; ４曲目演奏開始時間(分)
    mov ah,[esi+16]  ; ４曲目演奏開始時間(秒)
    mov al,[esi+17]  ; ４曲目演奏開始時間(FRAME)
    jmp short Calc_Music_END_Time

Disc_Time:
    mov dl,[esi+03]  ; ﾃﾞｨｽｸ内時間(分)
    mov ah,[esi+04]  ; ﾃﾞｨｽｸ内時間(秒)
    mov al,[esi+05]  ; ﾃﾞｨｽｸ内時間(FRAME)

Calc_Music_END_Time:
    call conv_msf2lsn ; 分:秒:FRAMEを論理ｾｸﾀ番号に変換
    dec ax       ; ３曲目の最後を計算する
    sbb dx,0
    call conv_lsn2msf ; 論理ｾｸﾀ番号を分:秒:FRAMEに変換

    mov [edi+03],dl  ; ３曲目演奏終了時間(分)
    mov [edi+04],ah  ; ３曲目演奏終了時間(秒)
    mov [edi+05],al  ; ３曲目演奏終了時間(FRAME)
    mov ax,250Fh ; 386ﾈｲﾃｨﾌﾞﾓｰﾄﾞｱﾄﾞﾚｽのMS-DOSｱﾄﾞﾚｽへの変換
    lea ebx,PLAY3    ; 変換する386ﾈｲﾃｨﾌﾞﾓｰﾄﾞｱﾄﾞﾚｽ
    mov ecx,6        ; ﾃﾞｰﾀ長
    int 21h

    jc  error

    mov di,cx
    ror ecx,16
    mov DSWD,cx
```

```
    mov ah,50h        ; 音楽演奏ｽﾀｰﾄ
    mov al,DEVNO ; ﾃﾞﾊﾞｲｽ番号
    mov ch,00h
    mov cl,01h        ; 時間指定

    mov EAXDWD,eax
    mov bx,93h
    mov RINT,bx       ; 割り込み番号

    mov ax,2511h ; ﾘｱﾙﾓｰﾄﾞ割り込みの発行
    lea edx,PARABLK   ; ﾊﾟﾗﾒｰﾀﾌﾞﾛｯｸのﾎﾟｲﾝﾀ
    int 21h

check:
    mov ah,53h        ;音楽演奏チェック
    mov al,DEVNO
    mov ch,00
    mov cl,00
    int 93h

    cmp al,1      ;演奏中ならcheckへ戻る
    jz  check

stop:
    mov ah,52h        ;CD演奏停止
    mov al,DEVNO
    mov ch,00
    int 93h
    jmp short owari
;
;   End Process
;
owari:
    mov ax,4c00h
    int 21h

;
;   エラー処理
;
error:

    ;
    ;   各エラー処理
    ;

    jmp short owari

;+-----------------------------------------------------------------+
;   分:秒:FRAME を 論理ｾｸﾀ番号 に変換
;+-----------------------------------------------------------------+
;
; 論理ｾｸﾀ番号 = ((( 分 * 60 + 秒 ) * 75 ) + FRAME ) - 150
;
conv_msf2lsn:
        push    bx
        push    cx

        mov     bx,ax
        mov     al,dl
        mov     cl,60
        mul     cl
        add     al,bh
        adc     ah,0
        mov     cx,75
        mul     cx
        mov     bh,0
        add     ax,bx
        adc     dx,0
        sub     ax,150
        sbb     dx,0

        pop     cx
    pop bx
    ret

 Page
;+-----------------------------------------------------------------+
;   論理ｾｸﾀ番号 を 分:秒:FRAME に 変換
;+-----------------------------------------------------------------+
;   分    =  ( 論理ｾｸﾀ番号 + 150 ) / ( 75 * 60 )
;   秒    = (( 論理ｾｸﾀ番号 + 150 ) - ( 秒 * 60 * 75 )) / 75
;   FRAME =  ( 論理ｾｸﾀ番号 + 150 ) % 75
;
conv_lsn2msf:
        push    cx

        add     ax,150
        adc     dx,0
        mov     cx,75*60
        div     cx
        xchg    dx,ax
        mov     cl,75
        div     cl
        xchg    ah,al
        xor     dh,dh

        pop     cx
        ret

code ends

;
;   データ領域
;

data    segment dword 'DATA' use32 ; 32bit データ宣言

TOCDATA  db  303 DUP(0)
PLAY3    db  6 DUP(0)

PARABLK  label    far
RINT dw  0
DSWD dw  0
ESWD dw  0
FSWD dw  0
GSWD dw  0
EAXDWD   dd  0
EDXDWD   dd  0

data ends
    end
```

B.2　描画プログラム

このプログラムでは，EGBの機能コード40H以降のファンクションを使用していますので，次のような式でスタックを確保する必要があります．

スタックサイズ＝仮想画面サイズ／8
＝1024×512／8
＝65536バイト

さらに，このプログラムを単独で実行するには，6KBのテンポラリスタックを加える必要があります．

必要なスタックサイズは，次の式のようになります．

スタックサイズ＝65536＋6144
＝71680

よって，リンク時に，386linkのリンクオプションで次のように指定してください．

```
A＞386link　プログラム名　-stack　71680
```

```
        .386p                   ; 386プロテクト宣言

DosInt       equ 021h          ;
DosRet       equ 04c00h            ;
NoError      equ 0             ;

Tbios        equ 0110h
EgbWorkSize  equ 1536

EgbInit      equ 000h          ; 初期化
EgbDisplayPage    equ 006h         ; 表示ページの指定
EgbRectangle equ 046h          ; 矩形
EgbCircle    equ 047h          ; 円

code    segment dword 'CODE' use32 ; 32bit コード宣言
        assume   cs:code,ds:data
                           ; コード領域に code
                           ; データ領域に data
                           ;   セグメントを指定
$EgbCall macro    FUNC
        mov ah,FUNC
        call pword ptr fs:[020h]   ; EGB bios call
        endm

        mov ax,Tbios
        mov fs,ax

        mov ecx,EgbWorkSize
        mov edi,offset EgbWork
        $EgbCall EgbInit       ; 初期化
                           ;   初期値として,
                           ;   640*480 2画面 16色に設定
        mov al,0
        mov edx,3
        $EgbCall EgbDisplayPage    ; 表示ページの設定
                           ;   解像度の設定
                           ;   書き込みページの指定
                           ;   描画モード
                           ;   面塗りモード
                           ;   描画色は初期値を使用
        mov esi,offset Circle ; 円パラメタアドレス
        $EgbCall EgbCircle    ; 円描画

        mov esi,offset Rectangle  ; 矩形パラメタアドレス
        $EgbCall EgbRectangle ; 矩形描画

        mov ax,DosRet+NoError
        int DosInt

code    ends

data    segment dword 'DATA' use32 ; 32bit データ宣言
EgbWork     db  EgbWorkSize dup(?)

Circle      dw  250           ; 中心点X座標
        dw  150           ; 中心点Y座標
        dw  100           ; 半径

Rectangle   dw  250           ; 始点X座標
        dw  200           ; 始点Y座標
        dw  490           ; 終点X座標
        dw  400           ; 終点Y座標

data    ends

        end
```

付録C

ネイティブBIOSのサンプルプログラム

ここでは，ネイティブBIOSを利用するプログラムのサンプルとして，以下のようにBIOSの種類ごとにアセンブラソースを掲載します．これらのプログラムはC言語のライブラリの形式をとっています．

ユーザープログラムを作成する際の参考にしてください．

- 共通ファイルサンプル
- グラフィックBIOSサンプル
- スプライトBIOSサンプル
- マウスBIOSサンプル
- フォントBIOSサンプル
- サウンドBIOSサンプル
- システム情報BIOSサンプル
- 拡張サウンドBIOSサンプル
- 音源割り込み管理BIOSサンプル

C.1　共通ファイルサンプル

vector.h

```
;------------------------------------------------------------------------------
;
;       ＴＢＩＯＳ　Ｃａｌｌベクタ
;
;------------------------------------------------------------------------------

ROM_CSEG                equ     0110h
ROM_DSEG                equ     0118h

EGB_OFFSET              equ     0020h
EGB_SEGMENT             equ     0024h

MOS_OFFSET              equ     0040h
MOS_SEGMENT             equ     0044h
MOSINT_OFFSET           equ     0048h
MOSINT_SEGMENT          equ     004ch
MOSIO_OFFSET            equ     0050h
MOSIO_SEGMENT           equ     0054h
MOSWORK_OFFSET          equ     0058h
MOSWORK_SEGMENT         equ     005ch

SPR_OFFSET              equ     0060h
SPR_SEGMENT             equ     0064h

SND_OFFSET              equ     0080h
SND_SEGMENT             equ     0084h
SNDWORK_OFFSET          equ     0098h
SNDWORK_SEGMENT         equ     009ch

FNT_OFFSET              equ     00a0h
FNT_SEGMENT             equ     00a4h

EUP_OFFSET              equ     00c0h
EUP_SEGMENT             equ     00c4h
EUPINT_OFFSET           equ     00c8h
EUPINT_SEGMENT          equ     00cch

INT_OFFSET              equ     01a0h
INT_SEGMENT             equ     01a4h
INTERVAL                equ     01a8h

SYS_OFFSET              equ     01c0h
SYS_SEGMENT             equ     01c4h
```

toolbox.h

```
;------------------------------------------------------------------------------
;
;       TownsBIOS assembler source list short form utility
;
;------------------------------------------------------------------------------
SMOV            macro   REG,PARA
                push    PARA                    ; mov REG,PARA
                pop     REG                     ; ３バイトで表現
                endm

calld           macro   data
                call    dword ptr data
                endm

callp           macro   data
                call    pword ptr data
                endm

movmd           macro   data,reg
                mov     dword ptr data,reg
                endm

movmw           macro   data,reg
                mov     word ptr data,reg
                endm

movmb           macro   data,reg
                mov     byte ptr data,reg
                endm

movdm           macro   reg,data
                mov     reg,dword ptr data
                endm

movwm           macro   reg,data
                mov     reg,word ptr data
                endm

movbm           macro   reg,data
                mov     reg,byte ptr data
                endm

movmo           macro   data,reg
                mov     data,offset reg
                endm

movro           macro   reg,data
                mov     reg,offset data
                endm
```

```
cmpd            macro   data,reg
                cmp     dword ptr data,reg
                endm

cmpw            macro   data,reg
                cmp     word ptr data,reg
                endm

cmpb            macro   data,reg
                cmp     byte ptr data,reg
                endm

testd           macro   data,reg
                test    dword ptr data,reg
                endm

testw           macro   data,reg
                test    word ptr data,reg
                endm

testb           macro   data,reg
                test    byte ptr data,reg
                endm

ord             macro   data,reg
                or      dword ptr data,reg
                endm

orw             macro   data,reg
                or      word ptr data,reg
                endm

orb             macro   data,reg
                or      byte ptr data,reg
                endm

andd            macro   data,reg
                and     dword ptr data,reg
                endm

andw            macro   data,reg
                and     word ptr data,reg
                endm

andb            macro   data,reg
                and     byte ptr data,reg
                endm

xord            macro   data,reg
                xor     dword ptr data,reg
                endm

xorw            macro   data,reg
                xor     word ptr data,reg
                endm

xorb            macro   data,reg
                xor     byte ptr data,reg
                endm

movzxb          macro   reg,data
                movzx   reg,byte ptr data
                endm

movzxw          macro   reg,data
                movzx   reg,word ptr data
                endm
```

tbioslib.h

```
;-----------------------------------------------------------------------------
;
;       ＴＢＩＯＳライブラリ　定数情報＆マクロ
;
;-----------------------------------------------------------------------------
include vector.h
include toolbox.h
;-----------------------------------------------------------------------------
INT_TYPE_SOUND  equ     04dh            ; FM/PCM interval vector number
TBIOS_B_COUNT   equ     0b0h            ; TBIOS mouse counter (23.04ms)
PAT             equ     0

@CMOS_count     equ     03b04h
@CMOS_moscfg    equ     03b06h
@CMOS_palette   equ     03b0ah
@CMOS_drive     equ     03c32h
@CMOS_real      equ     03ad6h
@real4          equ     0d4h
@real2          equ     0d2h
@real0          equ     0d0h

@def_click      equ     03eh
@def_mickey     equ     008h
@count          equ     004h
@moscfg         equ     006h

DTA_length      equ     44
PATH_length     equ     128
file_max        equ     4096
palette_max     equ     8*16+4
palette_all     equ     16
tmenu_color     equ     10
AKIDATA_size    equ     32-12
DEFDATA_size    equ     12
```

```
DATA_size       equ     32
DEF_MAX         equ     9
USER_MIN        equ     14
USER_MAX        equ     15
REAL_SEG        equ     060h
PARA_offset     equ     256+8
;-----------------------------------------------------------------------------
$biosCall       macro   rom,data
                mov     ah,data
                push    es
                push    dword ptr ROM_CSEG
                pop     es
                call    pword ptr es:[rom]
                pop     es
                endm

$biosCall2      macro   rom
                call    pword ptr es:[rom]
                endm

$biosCall3      macro   rom,data
                mov     ah,data
                push    es
                push    gs
                push    dword ptr ROM_CSEG
                pop     es
                push    ds
                pop     gs
                call    pword ptr es:[rom]
                pop     gs
                pop     es
                endm

$biosCall4      macro   rom,data
                mov     ah,data
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[rom]
                pop     fs
                endm

$biosCall5      macro   rom,data
                mov     ah,data
                push    fs
                push    gs
                push    dword ptr ROM_CSEG
                pop     fs
                push    ds
                pop     gs
                call    pword ptr fs:[rom]
                pop     gs
                pop     fs
                endm

$biosCall6      macro   rom
                push    es
                push    dword ptr ROM_CSEG
                pop     es
                call    pword ptr es:[rom]
                pop     es
                endm
```

sndlib.h

```
;-----------------------------------------------------------------------------
;
;       ＳＮＤライブラリ　定数情報＆マクロ
;
;-----------------------------------------------------------------------------
INT_TYPE_SOUND  equ     4dh             ; interrupt level
TIMER_B_SND     equ     0ddh            ; SND mouse counter (10.08ms)
TBIOS_B_MOS     equ     0b0h            ; MOS mouse counter (23.04ms)
TIMER_A_COUNT   equ     001h            ; default count
;-----------------------------------------------------------------------------
include toolbox.h
include vector.h
;-----------------------------------------------------------------------------

$biosCall       macro   rom,data
                mov     ah,data
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[rom]
                pop     fs
                endm

$biosCall2      macro   rom,data
                mov     ah,data
                call    pword ptr fs:[rom]
                endm
```

codebgn.mac

```
page    ,132
                .386p
                assume  cs:TBIOSLIB_code
CGROUP          group   TBIOSLIB_code
TBIOSLIB_code   segment dword public 'CODE' use32
                include tbioslib.h
```

codeend.mac

```
TBIOSLIB_code   ends
end
```

databgn.mac

```
page    ,132
                .386p
                assume  ds:TBIOSLIB_data
DGROUP          group   TBIOSLIB_data
TBIOSLIB_data   segment dword public 'DATA' use32
                include tbioslib.h
```

dataend.mac

```
TBIOSLIB_data   ends
end
```

C.2　グラフィックBIOSサンプル

egb_00.asm

```
include codebgn.mac

                public  EGB_init
EGB_init        proc    near
                push    gs
                push    edi
                mov     edi,12+PAT[esp]
                mov     ecx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,00h
                movsx   eax,ah
                pop     edi
                pop     gs
                ret
EGB_init        endp

include codeend.mac
```

egb_01.asm

```
include codebgn.mac

                public  EGB_resolution
EGB_resolution  proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                mov     dx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,01h
                movsx   eax,ah
                pop     edi
                ret
EGB_resolution  endp

include codeend.mac
```

egb_01a.asm

```
include codebgn.mac

                public  EGB_resolutionRam
EGB_resolutionRam       proc    near
                push    ebx
                push    esi
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     cx,24+PAT[esp]
                mov     dx,28+PAT[esp]
                mov     bx,32+PAT[esp]
                mov     esi,36+PAT[esp]
                $biosCall3      EGB_OFFSET,01h
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_resolutionRam       endp

include codeend.mac
```

egb_02.asm

```
include codebgn.mac

                public  EGB_displayStart
EGB_displayStart        proc    near
                push    ebx
```

```
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     dx,20+PAT[esp]
                mov     bx,24+PAT[esp]
                $biosCall3      EGB_OFFSET,02h
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
EGB_displayStart        endp

include codeend.mac
```

egb_03.asm

```
include codebgn.mac

                public  EGB_viewport
EGB_viewport    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,03h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_viewport    endp

include codeend.mac
```

egb_04.asm

```
include codebgn.mac

                public  EGB_palette
EGB_palette     proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,04h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_palette     endp
```

```
include codeend.mac
```

egb_05.asm

```
include codebgn.mac

                public  EGB_writePage
EGB_writePage   proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,05h
                movsx   eax,ah
                pop     edi
                ret
EGB_writePage   endp

include codeend.mac
```

egb_06.asm

```
include codebgn.mac

                public  EGB_displayPage
EGB_displayPage proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                mov     edx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,06h
                movsx   eax,ah
                pop     edi
                ret
EGB_displayPage endp

include codeend.mac
```

egb_07.asm

```
include codebgn.mac

                public  EGB_color
EGB_color       proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                mov     edx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,07h
                movsx   eax,ah
                pop     edi
```

```
                ret
EGB_color       endp

include codeend.mac
```

egb_08.asm

```
include codebgn.mac

                public  EGB_colorIGRB
EGB_colorIGRB   proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                mov     edx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,08h
                movsx   eax,ah
                pop     edi
                ret
EGB_colorIGRB   endp

include codeend.mac
```

egb_09.asm

```
include codebgn.mac

                public  EGB_pastel
EGB_pastel      proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     dx,12+PAT[esp]
                $biosCall3      EGB_OFFSET,09h
                movsx   eax,ah
                pop     edi
                ret
EGB_pastel      endp

include codeend.mac
```

egb_0a.asm

```
include codebgn.mac

                public  EGB_writeMode
EGB_writeMode   proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,0ah
                movsx   eax,ah
                pop     edi
                ret
EGB_writeMode   endp

include codeend.mac
```

egb_0b.asm

```
include codebgn.mac

                public  EGB_linePattern
EGB_linePattern proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                mov     edx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,0bh
                movsx   eax,ah
                pop     edi
                ret
EGB_linePattern endp

include codeend.mac
```

egb_0c.asm

```
include codebgn.mac

                public  EGB_paintMode
EGB_paintMode   proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     dx,12+PAT[esp]
                $biosCall3      EGB_OFFSET,0ch
                movsx   eax,ah
                pop     edi
                ret
EGB_paintMode   endp

include codeend.mac
```

egb_0d.asm

```
include codebgn.mac

                public  EGB_hatchingPattern
EGB_hatchingPattern     proc    near
                push    ebx
                push    esi
```

```
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     bh,24+PAT[esp]
                mov     bl,28+PAT[esp]
                mov     esi,32+PAT[esp]
                $biosCall3      EGB_OFFSET,0dh
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_hatchingPattern     endp

include codeend.mac
```

egb_0e.asm

```
include codebgn.mac

                public  EGB_tilePattern
EGB_tilePattern proc    near
                push    ebx
                push    esi
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     bh,24+PAT[esp]
                mov     bl,28+PAT[esp]
                mov     esi,32+PAT[esp]
                $biosCall3      EGB_OFFSET,0eh
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_tilePattern endp

include codeend.mac
```

egb_0f.asm

```
include codebgn.mac

                public  EGB_maskRegion
EGB_maskRegion  proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,0fh
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_maskRegion  endp

include codeend.mac
```

egb_10.asm

```
include codebgn.mac

                public  EGB_mask
EGB_mask        proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,10h
                movsx   eax,ah
                pop     edi
                ret
EGB_mask        endp

include codeend.mac
```

egb_11.asm

```
include codebgn.mac

                public  EGB_pen
EGB_pen         proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,11h
                movsx   eax,ah
                pop     edi
                ret
EGB_pen         endp

include codeend.mac
```

egb_12.asm

```
include codebgn.mac

                public  EGB_penSize
EGB_penSize     proc    near
                push    edi
                mov     edi,8+PAT[esp]
```

```
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,12h
                movsx   eax,ah
                pop     edi
                ret
EGB_penSize     endp

include codeend.mac
```

egb_13.asm

```
include codebgn.mac

                public  EGB_penStyle
EGB_penStyle    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,13h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_penStyle    endp

include codeend.mac
```

egb_14.asm

```
include codebgn.mac

                public  EGB_maskBit
EGB_maskBit     proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     edx,12+PAT[esp]
                $biosCall3      EGB_OFFSET,14h
                movsx   eax,ah
                pop     edi
                ret
EGB_maskBit     endp

include codeend.mac
```

egb_15.asm

```
include codebgn.mac

                public  EGB_textDirection
EGB_textDirection       proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,15h
                movsx   eax,ah
                pop     edi
                ret
EGB_textDirection       endp

include codeend.mac
```

egb_16.asm

```
include codebgn.mac

                public  EGB_textDisplayDirection
EGB_textDisplayDirection        proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,16h
                movsx   eax,ah
                pop     edi
                ret
EGB_textDisplayDirection        endp

include codeend.mac
```

egb_17.asm

```
include codebgn.mac

                public  EGB_textSpace
EGB_textSpace   proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     dx,12+PAT[esp]
                $biosCall3      EGB_OFFSET,17h
                movsx   eax,ah
                pop     edi
                ret
EGB_textSpace   endp

include codeend.mac
```

egb_18.asm

```
include codebgn.mac
```

```
                public  EGB_textZoom
EGB_textZoom    proc    near
                push    ebx
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     dx,20+PAT[esp]
                mov     bx,24+PAT[esp]
                $biosCall3      EGB_OFFSET,18h
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
EGB_textZoom    endp

include codeend.mac
```

egb_19.asm

```
include codebgn.mac

                public  EGB_fontStyle
EGB_fontStyle   proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     dx,12+PAT[esp]
                $biosCall3      EGB_OFFSET,19h
                movsx   eax,ah
                pop     edi
                ret
EGB_fontStyle   endp

include codeend.mac
```

egb_1a.asm

```
include codebgn.mac

                public  EGB_superImpose
EGB_superImpose proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,1ah
                movsx   eax,ah
                pop     edi
                ret
EGB_superImpose endp

include codeend.mac
```

egb_1b.asm

```
include codebgn.mac

                public  EGB_dezitize
                public  EGB_digitize
EGB_dezitize    proc    near
EGB_digitize:
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                $biosCall3      EGB_OFFSET,1bh
                movsx   eax,ah
                pop     edi
                ret
EGB_dezitize    endp

include codeend.mac
```

egb_1c.asm

```
include codebgn.mac

                public  EGB_resolutionHandle
EGB_resolutionHandle    proc    near
                push    edi
                mov     edi,8+PAT[esp]
                mov     al,12+PAT[esp]
                mov     dx,16+PAT[esp]
                $biosCall3      EGB_OFFSET,1ch
                movsx   eax,ah
                pop     edi
                ret
EGB_resolutionHandle    endp

include codeend.mac
```

egb_1da.asm

```
include codebgn.mac

                public  EGB_getStackSize
EGB_getStackSize        proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                mov     ax,1d00h
                $biosCall6      EGB_OFFSET
                movsx   eax,ah
```

```
		pop	edi
		pop	esi
		ret
EGB_getStackSize	endp

include codeend.mac
```

egb_1db.asm

```
include codebgn.mac

		public	EGB_setStackAddress
EGB_setStackAddress	proc	near
		push	esi
		push	edi
		mov	edi,12+PAT[esp]
		mov	esi,16+PAT[esp]
		mov	ax,1d01h
		$biosCall16	EGB_OFFSET
		movsx	eax,ah
		pop	edi
		pop	esi
		ret
EGB_setStackAddress	endp

include codeend.mac
```

egb_1dc.asm

```
include codebgn.mac

		extrn	_egbStackAlloc	:word
		extrn	_egbStackFree	:word

		public	EGB_setStackEvent
EGB_setStackEvent	proc	near
		push	esi
		push	edi
		mov	edi,12+PAT[esp]
		mov	ecx,16+PAT[esp]
		mov	edx,20+PAT[esp]
		mov	esi,offset _egbStackAlloc
		mov	ds:[esi+00],ecx
		mov	ds:[esi+04],ds
		mov	ds:[esi+06],es
		mov	ds:[esi+08],fs
		mov	ds:[esi+10],gs
		mov	esi,offset _egbStackFree
		mov	ds:[esi+00],edx
		mov	ds:[esi+04],ds
		mov	ds:[esi+06],es
		mov	ds:[esi+08],fs
		mov	ds:[esi+10],gs
		sub	eax,eax
		mov	ax,cs
		push	eax
		push	offset EGB_stackEventFree
		push	eax
		push	offset EGB_stackEventAlloc
		mov	esi,esp
		mov	ax,1d02h
		$biosCall16	EGB_OFFSET
		movsx	eax,ah
		add	esp,16
		pop	edi
		pop	esi
		ret
EGB_setStackEvent	endp

		align	4
EGB_stackEventAlloc	proc	far
		pushfd
		pushad
		push	ds
		push	es
		push	fs
		push	gs
		mov	esi,offset _egbStackAlloc
		mov	ds,cs:[esi+04]
		mov	es,cs:[esi+06]
		mov	fs,cs:[esi+08]
		mov	gs,cs:[esi+10]
		push	ecx
		call	dword ptr cs:[esi+00]
		pop	ecx
		pop	gs
		pop	fs
		pop	es
		pop	ds
		popad
		popfd
		ret
EGB_stackEventAlloc	endp

		align	4
EGB_stackEventFree	proc	far
		pushfd
		pushad
		push	ds
		push	es
		push	fs
		push	gs
		mov	esi,offset _egbStackFree
		mov	ds,cs:[esi+04]
```

```
                mov     es,cs:[esi+06]
                mov     fs,cs:[esi+08]
                mov     gs,cs:[esi+10]
                call    dword ptr cs:[esi+00]
                pop     gs
                pop     fs
                pop     es
                pop     ds
                popad
                popfd
                ret
EGB_stackEventFree      endp

include codeend.mac
```

egb_1dd.asm

```
include databgn.mac
;------------------------------------------------------------------------------
                public  _egbStackAlloc
                public  _egbStackFree
;------------------------------------------------------------------------------
_egbStackAlloc  dd      0                       ; イベントアドレス
                dw      0                       ; DS
                dw      0                       ; ES
                dw      0                       ; FS
                dw      0                       ; GS
_egbStackFree   dd      0                       ; イベントアドレス
                dw      0                       ; DS
                dw      0                       ; ES
                dw      0                       ; FS
                dw      0                       ; GS
;------------------------------------------------------------------------------
include dataend.mac
```

egb_1e.asm

```
include codebgn.mac

                public  EGB_digitizeAdjust
EGB_digitizeAdjust      proc    near
                push    ebx
                push    edi
                mov     edi,12+PAT[esp]
                mov     dx,16+PAT[esp]
                mov     bx,20+PAT[esp]
                $biosCall3      EGB_OFFSET,1eh
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
EGB_digitizeAdjust      endp

include codeend.mac
```

egb_20.asm

```
include codebgn.mac

                public  EGB_clearScreen
EGB_clearScreen proc    near
                push    edi
                mov     edi,8+PAT[esp]
                $biosCall3      EGB_OFFSET,20h
                movsx   eax,ah
                pop     edi
                ret
EGB_clearScreen endp

include codeend.mac
```

egb_21.asm

```
include codebgn.mac

                public  EGB_partClearScreen
EGB_partClearScreen     proc    near
                push    edi
                mov     edi,8+PAT[esp]
                $biosCall3      EGB_OFFSET,21h
                movsx   eax,ah
                pop     edi
                ret
EGB_partClearScreen     endp

include codeend.mac
```

egb_22.asm

```
include codebgn.mac

                public  EGB_getBlockColor
EGB_getBlockColor       proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,22h
                movsx   eax,ah
                pop     edi
                pop     esi
```

```
                ret
EGB_getBlockColor       endp

include codeend.mac
```

egb_23.asm

```
include codebgn.mac

                public  EGB_putBlockColor
EGB_putBlockColor       proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,23h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_putBlockColor       endp

include codeend.mac
```

egb_24.asm

```
include codebgn.mac

                public  EGB_getBlock
EGB_getBlock    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,24h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_getBlock    endp

include codeend.mac
```

egb_25.asm

```
include codebgn.mac

                public  EGB_putBlock
EGB_putBlock    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,25h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_putBlock    endp

include codeend.mac
```

egb_26.asm

```
include codebgn.mac

                public  EGB_getBlockZoom
EGB_getBlockZoom        proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,26h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_getBlockZoom        endp

include codeend.mac
```

egb_27.asm

```
include codebgn.mac

                public  EGB_putBlockZoom
EGB_putBlockZoom        proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,27h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_putBlockZoom        endp
```

```
include codeend.mac
```

egb_28.asm

```
include codebgn.mac

                public  EGB_graphicCursor
EGB_graphicCursor       proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,28h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_graphicCursor       endp

include codeend.mac
```

egb_29.asm

```
include codebgn.mac

                public  EGB_maskData
EGB_maskData    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,29h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_maskData    endp

include codeend.mac
```

egb_2a.asm

```
include codebgn.mac

                public  EGB_scroll
EGB_scroll      proc    near
                push    ebx
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     dx,20+PAT[esp]
                mov     bx,24+PAT[esp]
                $biosCall3      EGB_OFFSET,2ah
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
EGB_scroll      endp

include codeend.mac
```

egb_2b.asm

```
include codebgn.mac

                public  EGB_partScroll
EGB_partScroll  proc    near
                push    ebx
                push    esi
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     dx,24+PAT[esp]
                mov     bx,28+PAT[esp]
                mov     esi,32+PAT[esp]
                $biosCall3      EGB_OFFSET,2bh
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_partScroll  endp

include codeend.mac
```

egb_2c.asm

```
include codebgn.mac

                public  EGB_region
EGB_region      proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    esi
                push    edi
                mov     edi,[ebp+08]
                mov     eax,[ebp+12]
```

```
                mov     al,ds:[eax]
                mov     edx,[ebp+20]
                mov     dx,ds:[edx]
                mov     ebx,[ebp+24]
                mov     bx,ds:[ebx]
                mov     esi,[ebp+36]
                $biosCall3      EGB_OFFSET,2ch
                push    eax
                mov     eax,[ebp+20]
                movzx   edx,dx
                mov     ds:[eax],edx
                mov     eax,[ebp+24]
                movzx   ebx,bx
                mov     ds:[eax],ebx
                pop     eax
                mov     ebx,[ebp+28]
                movzx   esi,si
                mov     ds:[ebx],esi
                mov     ebx,[ebp+32]
                movzx   edi,di
                mov     ds:[ebx],edi
                mov     ebx,[ebp+12]
                movzx   edx,al
                mov     ds:[ebx],edx
                mov     ebx,[ebp+16]
                mov     ds:[ebx],ecx
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                leave
                ret
EGB_region      endp

include codeend.mac
```

egb_2d.asm

```
include codebgn.mac

                public  EGB_copy
EGB_copy        proc    near
                push    ebx
                push    esi
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     esi,24+PAT[esp]
                mov     ebx,28+PAT[esp]
                $biosCall5      EGB_OFFSET,2dh
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_copy        endp

include codeend.mac
```

egb_2e.asm

```
include codebgn.mac

                public  EGB_rotate
EGB_rotate      proc    near
                push    ebx
                push    esi
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     esi,24+PAT[esp]
                mov     ebx,28+PAT[esp]
                $biosCall5      EGB_OFFSET,2eh
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_rotate      endp

include codeend.mac
```

egb_2f.asm

```
include codebgn.mac

                public  EGB_resolve
EGB_resolve     proc    near
                push    ebx
                push    edi
                mov     edi,12+PAT[esp]
                mov     ebx,16+PAT[esp]
                $biosCall5      EGB_OFFSET,2fh
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
EGB_resolve     endp

include codeend.mac
```

egb_30.asm

```
include codebgn.mac

                public  EGB_copyBlock
EGB_copyBlock   proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,30h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_copyBlock   endp

include codeend.mac
```

egb_40.asm

```
include codebgn.mac

                public  EGB_pset
EGB_pset        proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,40h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_pset        endp

include codeend.mac
```

egb_41.asm

```
include codebgn.mac

                public  EGB_connect
EGB_connect     proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,41h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_connect     endp

include codeend.mac
```

egb_42.asm

```
include codebgn.mac

                public  EGB_unConnect
EGB_unConnect   proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,42h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_unConnect   endp

include codeend.mac
```

egb_43.asm

```
include codebgn.mac

                public  EGB_polygon
EGB_polygon     proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,43h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_polygon     endp

include codeend.mac
```

egb_44.asm

```
include codebgn.mac

                public  EGB_rotatePolygon
```

```
EGB_rotatePolygon       proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,44h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_rotatePolygon       endp

include codeend.mac
```

egb_45.asm

```
include codebgn.mac

                public  EGB_triangle
EGB_triangle    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,45h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_triangle    endp

include codeend.mac
```

egb_46.asm

```
include codebgn.mac

                public  EGB_rectangle
EGB_rectangle   proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,46h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_rectangle   endp

include codeend.mac
```

egb_47.asm

```
include codebgn.mac

                public  EGB_circle
EGB_circle      proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,47h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_circle      endp

include codeend.mac
```

egb_48.asm

```
include codebgn.mac

                public  EGB_arc
EGB_arc         proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,48h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_arc endp

include codeend.mac
```

egb_49.asm

```
include codebgn.mac

                public  EGB_fan
EGB_fan         proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,49h
                movsx   eax,ah
```

```
                pop     edi
                pop     esi
                ret
EGB_fan         endp

include codeend.mac
```

egb_4a.asm

```
include codebgn.mac

                public  EGB_ellipse
EGB_ellipse     proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,4ah
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_ellipse     endp

include codeend.mac
```

egb_4b.asm

```
include codebgn.mac

                public  EGB_ellipticArc
EGB_ellipticArc proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,4bh
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_ellipticArc endp

include codeend.mac
```

egb_4c.asm

```
include codebgn.mac

                public  EGB_ellipticFan
EGB_ellipticFan proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,4ch
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_ellipticFan endp

include codeend.mac
```

egb_4d.asm

```
include codebgn.mac

                public  EGB_paint
EGB_paint       proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,4dh
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_paint       endp

include codeend.mac
```

egb_4e.asm

```
include codebgn.mac

                public  EGB_closePaint
EGB_closePaint  proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,4eh
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_closePaint  endp

include codeend.mac
```

egb_4f.asm

```
include codebgn.mac

                public  EGB_point
EGB_point       proc    near
                push    ebx
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     dx,20+PAT[esp]
                mov     bx,24+PAT[esp]
                $biosCall3      EGB_OFFSET,4fh
                mov     ebx,28+PAT[esp]
                mov     [ebx],edx
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
EGB_point       endp

include codeend.mac
```

egb_50.asm

```
include codebgn.mac

                public  EGB_bow
EGB_bow         proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,50h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_bow         endp

include codeend.mac
```

egb_51.asm

```
include codebgn.mac

                public  EGB_semiBow
EGB_semiBow     proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,51h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_semiBow     endp

include codeend.mac
```

egb_60.asm

```
include codebgn.mac

                public  EGB_sjisString
EGB_sjisString  proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,60h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_sjisString  endp

include codeend.mac
```

egb_61.asm

```
include codebgn.mac

                public  EGB_connectSjisString
EGB_connectSjisString   proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,61h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_connectSjisString   endp

include codeend.mac
```

egb_62.asm

```
include codebgn.mac

                public  EGB_asciiString
EGB_asciiString proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     al,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall3      EGB_OFFSET,62h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_asciiString endp

include codeend.mac
```

egb_63.asm

```
include codebgn.mac

                public  EGB_connectAsciiString
EGB_connectAsciiString  proc    near
                push    ebx
                push    esi
                push    edi
                mov     edi,16+PAT[esp]
                mov     al,20+PAT[esp]
                mov     esi,24+PAT[esp]
                $biosCall3      EGB_OFFSET,63h
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
EGB_connectAsciiString  endp

include codeend.mac
```

egb_64.asm

```
include codebgn.mac

                public  EGB_jisString
EGB_jisString   proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,64h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_jisString   endp

include codeend.mac
```

egb_65.asm

```
include codebgn.mac

                public  EGB_connectJisString
EGB_connectJisString    proc    near
                push    esi
                push    edi
                mov     edi,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall3      EGB_OFFSET,65h
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
EGB_connectJisString    endp

include codeend.mac
```

egb_66.asm

```
include codebgn.mac

                public  EGB_anyChar
EGB_anyChar     proc    near
                push    esi
                push    edi
                push    ebx
                mov     edi,16+PAT[esp]
                mov     dx,20+PAT[esp]
                mov     bx,24+PAT[esp]
                mov     esi,28+PAT[esp]
                $biosCall3      EGB_OFFSET,66h
                movsx   eax,ah
                pop     ebx
                pop     edi
                pop     esi
                ret
EGB_anyChar     endp
```

```
include codeend.mac
```

C.3　スプライトBIOSサンプル

spr_00.asm

```
include codebgn.mac

;-----------------------------------------------------
;
;       SPR_init
;
;-----------------------------------------------------
;
                public  SPR_init
SPR_init        proc    near
                $biosCall       SPR_OFFSET,00H
                movsx   eax,ah
                ret
SPR_init        endp

include codeend.mac
```

spr_01.asm

```
include codebgn.mac

;---------------------------------------------------------
;
;       SPR_display
;
;---------------------------------------------------------
;
                public  SPR_display
SPR_display     proc    near
                mov     al,4+PAT[esp]
                mov     cx,8+PAT[esp]
                $biosCall       SPR_OFFSET,01H
                movsx   eax,ah
                ret
SPR_display     endp
```

```
include codeend.mac
```

spr_02.asm

```
include codebgn.mac

;-----------------------------------------------------------
;
;       SPR_define
;
;-----------------------------------------------------------
;
                public  SPR_define
SPR_define      proc    near
                push    esi
                mov     al,8+PAT[esp]
                mov     cx,12+PAT[esp]
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     esi,24+PAT[esp]
                $biosCall       SPR_OFFSET,02H
                movsx   eax,ah
                pop     esi
                ret
SPR_define      endp

include codeend.mac
```

spr_03.asm

```
include codebgn.mac

;------------------------------------------------------------
;
;       SPR_setPaletteBlock
;
;------------------------------------------------------------
;
                public  SPR_setPaletteBlock
SPR_setPaletteBlock     proc    near
                push    esi
                mov     cx,8+PAT[esp]
                mov     dx,12+PAT[esp]
                mov     esi,16+PAT[esp]
                $biosCall       SPR_OFFSET,03H
                movsx   eax,ah
                pop     esi
                ret
SPR_setPaletteBlock     endp

include codeend.mac
```

spr_04.asm

```
include codebgn.mac

;------------------------------------------------------------
;
;       SPR_setPosition
;
;------------------------------------------------------------
;
                public  SPR_setPosition
SPR_setPosition proc    near
                push    esi
                push    edi
                mov     al,12+PAT[esp]
                mov     cx,16+PAT[esp]
                mov     dh,20+PAT[esp]
                mov     dl,24+PAT[esp]
                mov     si,28+PAT[esp]
                mov     di,32+PAT[esp]
                $biosCall       SPR_OFFSET,04H
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
SPR_setPosition endp

include codeend.mac
```

spr_05.asm

```
include codebgn.mac

;------------------------------------------------------------
;
;       SPR_setAttribute
;
;------------------------------------------------------------
;
                public  SPR_setAttribute
SPR_setAttribute proc   near
                push    esi
                push    edi
                mov     cx,12+PAT[esp]
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     si,24+PAT[esp]
                mov     di,28+PAT[esp]
                $biosCall       SPR_OFFSET,05H
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
SPR_setAttribute endp

include codeend.mac
```

spr_06.asm

```
include codebgn.mac

;------------------------------------------------------------
;
;       SPR_setMotion
;
;------------------------------------------------------------
;
                public  SPR_setMotion
SPR_setMotion   proc    near
                push    esi
                push    edi
                mov     cx,12+PAT[esp]
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     si,24+PAT[esp]
                mov     di,28+PAT[esp]
                $biosCall       SPR_OFFSET,06H
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
SPR_setMotion   endp

include codeend.mac
```

spr_07.asm

```
include codebgn.mac

;------------------------------------------------------------
;
;       SPR_setoffset
;
;------------------------------------------------------------
;
                public  SPR_setOffset
SPR_setOffset   proc    near
                push    esi
                push    edi
                mov     si,12+PAT[esp]
                mov     di,16+PAT[esp]
                $biosCall       SPR_OFFSET,07H
```

```
                movsx   eax,ah
                pop     edi
                pop     esi
                ret
SPR_setOffset   endp

include codeend.mac
```

spr_08.asm

```
include codebgn.mac

;-------------------------------------------------------
;
;       SPR_readAttribute
;
;-------------------------------------------------------
;
                public  SPR_readAttribute
SPR_readAttribute proc  near
                push    edi
                mov     cx,8+PAT[esp]
                mov     dh,12+PAT[esp]
                mov     dl,16+PAT[esp]
                mov     edi,20+PAT[esp]
                $biosCall4      SPR_OFFSET,08H
                movsx   eax,ah
                pop     edi
                ret
SPR_readAttribute endp

include codeend.mac
```

C.4　マウスBIOSサンプル

mos_00.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;
;       mouse library for Townes
;
;-----------------------------------------------------------------------------
                extrn   skb_event_adr   :dword
                extrn   mos_event_adr   :dword
                extrn   mos_control_flg :dword
                extrn   moswork_adr     :dword
                extrn   moswork_seg     :word
                extrn   save_mosdisp    :dword
                extrn   save_mosevent   :dword
                extrn   save_skbevent   :dword
                extrn   TBIOSSTK_bottom :byte
;-----------------------------------------------------------------------------

                db      "TOWNS MOUSE LIBRARY by Y.Miyazawa "
                db      "COPYRIGHT (C) 1993 FUJITSU LIMITED"

;-----------------------------------------------------------------------------
;       マウスインタフェースの動作開始
;       MOS_start()
;-----------------------------------------------------------------------------
                align   4
                public  MOS_start
MOS_start       proc    near
                push    ebp                     ;
                mov     ebp,esp                 ;
                push    edi                     ;
                push    fs                      ;
                push    gs                      ;
                pushfd                          ;
                push    dword ptr ROM_CSEG      ;
                pop     fs                      ; fs = TBIOS code segment
                push    ds                      ;
                pop     gs                      ; gs = ds
#version_check:                                 ;
                sub     eax,eax                 ;
                cmp     fs:[INT_OFFSET],eax     ;
                jz      short #error            ; TBIOS-V1.1 では動作させない
#work_init:                                     ;
                mov     ds:skb_event_adr,eax    ; ワークの初期化
                mov     ds:mos_event_adr,eax    ;
                mov     al,01111111b            ;
                mov     ds:mos_control_flg,eax  ;
#set_mosinit_para:                              ;
                mov     edi,ss:[ebp+8]          ; edi = mouse work
                mov     ecx,ss:[ebp+12]         ; ecx = work length
                mov     ds:moswork_seg,ds       ;
                mov     ds:moswork_adr,edi      ;
                mov     ah,1ah                  ;
                call    sysmain                 ; 動作状態の読み取り
                test    al,00000001b            ;
                jz      short #mouse_init       ; マウスＢＩＯＳ非動作？
                call    sidework_init           ; サイドワーク用初期化
                jmp     short #exit             ;
#mouse_init:                                    ;
```

```
		call	mouse_init		; 通常の初期化
#exit:						;
		popfd				;
		pop	gs			;
		pop	fs			;
		pop	edi			;
		leave				;
		ret				;
#error:						;
		dec	eax			;
		jmp	short #exit		;
MOS_start	endp
;-------------------------------------------------------------------------------
;	サイドワーク用初期化処理
;-------------------------------------------------------------------------------
		align	4
sidework_init	proc	near
		push	esi			;
		mov	ah,11h			;
		call	sysmain			; 表示／消去状態読取
		mov	ds:save_mosdisp,eax	;
		mov	ax,0200h		;
		call	mosmain			; マウスを消す
		cli				;
		mov	ax,1000h		;
		call	sysmain			; 親ワークの退避
		movro	esi,save_mosevent	;
		mov	ax,0803h		;
		call	intmain			; マウスイベント退避
		movro	esi,save_skbevent	;
		mov	ax,0804h		;
		call	intmain			; ＳＫＢイベント退避
		mov	ax,0703h		;
		call	intmain			; マウスイベント停止
		mov	ax,0704h		;
		call	intmain			; ＳＫＢイベント停止
		pop	esi			;
		ret				;
sidework_init	endp
;-------------------------------------------------------------------------------
;	初期化と割り込みハンドラの登録
;-------------------------------------------------------------------------------
		align	4
mouse_init	proc	near
		mov	ah,0			;
		call	mosmain			; マウスＢＩＯＳ動作開始
		test	ah,ah			;
		jz	short #set_config	; 正常終了？
		sub	eax,eax			;
		dec	eax			;
		ret				; エラー
#set_config:					;
		mov	dx,@CMOS_count		;
		in	al,dx			;
		movzx	edx,al			;
		test	edx,edx			; ダブルクリック間隔≠０？
		jnz	short #set_click	;
		mov	dl,@def_click		; 設定値が無い場合の処理
#set_click:					;
		shl	edx,4			; ×16倍
		mov	ax,3005h		;
		call	sysmain			; ダブルクリック間隔設定
		mov	dx,@CMOS_moscfg		;
		in	al,dx			;
		test	al,al			; パルス数／画素比≠０？
		jnz	short #set_mickey	;
		mov	al,@def_mickey		; 設定値が無い場合の処理
#set_mickey:					;
		mov	dl,al			;
		and	dl,01111111b		; Y &= 0x7f
		mov	dh,dl			; X = Y
		shr	al,7			;
		mov	ah,13h			;
		call	mosmain			; マウス左右入れ換えの設定
		mov	ah,0ch			;
		call	mosmain			; パルス数／画素比の設定
		cli				;
		movro	edx,TBIOSSTK_bottom	;
		mov	ah,01h			;
		call	intmain			; ＭＯＳ割り込み処理の登録
		test	eax,eax			;
		jnz	short #set_extender	; ＳＮＤ割り込み処理が未登録？
		ret				;
		align	4			;
#set_extender:					;
		push	ds			;
		push	es			;
		push	ebx			;
		mov	cl,INT_TYPE_SOUND	;
		mov	ax,2502h		;
		int	21h			; native mode vector read
		jc	short #error		;
		mov	edx,ebx			;
		mov	ax,3001h		;
		call	sysmain			; save native offset
		sub	edx,edx			;
		mov	dx,es			;
		mov	ax,3002h		;
		call	sysmain			; save native segment
		mov	cl,INT_TYPE_SOUND	;
		mov	ax,2503h		;
		int	21h			; real mode vector read
		jc	short #error		;
		movzx	edx,bx			;
		mov	ax,3003h		;
```

```
                call    sysmain                 ; save real offset
                shr     ebx,16                  ;
                mov     edx,ebx                 ;
                mov     ax,3004h                ;
                call    sysmain                 ; save real segment
                mov     cl,INT_TYPE_SOUND       ;
                lds     edx,fs:[INTERVAL]       ;
                mov     ax,2506h                ;
                int     21h                     ; 割り込みベクタの登録
                jc      short #error            ;
                mov     bl,0ffh                 ;
                mov     cx,TBIOS_B_COUNT        ;
                mov     ah,16h                  ;
                callp   fs:[SND_OFFSET]         ; Timer-B Set & Start
                sub     eax,eax                 ;
#exit:                                          ;
                pop     ebx                     ;
                pop     es                      ;
                pop     ds                      ;
                ret                             ;
#error:                                         ;
                sub     eax,eax                 ;
                dec     eax                     ;
                jmp     short #exit             ;
mouse_init      endp
;-----------------------------------------------------------------------------
;       マウスインタフェースの動作終了
;       MOS_end()
;-----------------------------------------------------------------------------
                align   4
                public  MOS_end
MOS_end         proc    near                    ;
                push    fs                      ;
                push    edi                     ;
                pushfd                          ;
                push    dword ptr ROM_CSEG      ; fs = TBIOS code segment
                pop     fs                      ;
                mov     edi,ds:moswork_adr      ;
                mov     ax,ds:moswork_seg       ;
                cmp     fs:[MOSWORK_OFFSET],edi ;
                jne     short #sidework         ;
                cmp     fs:[MOSWORK_SEGMENT],ax ;
                jne     short #sidework         ;
                call    mouse_end               ; マウス終了
                jmp     short #exit             ;
#sidework:                                      ;
                call    sidework_end            ; サイドワーク終了
#exit:                                          ;
                popfd                           ;
                pop     edi                     ;
                pop     fs                      ;
                ret                             ;
MOS_end         endp
```

```
;-----------------------------------------------------------------------------
;       サイドワーク用終了処理
;-----------------------------------------------------------------------------
                align   4
sidework_end    proc    near
                push    gs                      ;
                push    esi                     ;
                mov     gs,ax                   ;
                mov     ax,0200h                ;
                call    mosmain                 ; マウスを消す
                cli                             ;
                mov     ecx,4096                ;
                mov     ax,1001h                ;
                call    sysmain                 ; 親ワークの復元
                movro   esi,save_mosevent       ;
                mov     ax,0603h                ;
                call    intmain                 ; マウスイベントの復元
                movro   esi,save_skbevent       ;
                mov     ax,0604h                ;
                call    intmain                 ; ＳＫＢイベントの復元
                mov     edx,ds:save_mosdisp     ;
                mov     al,dl                   ; 表示状態
                shr     edx,16                  ; 表示レベル
                sub     ecx,ecx                 ; レベルカウンタクリア
                test    al,1                    ;
                jz      short #level_down       ;
                mov     eax,eax                 ; align 4
#level_up:                                      ;
                mov     ax,0203h                ; マウスレベル調整（正方向）
                call    mosmain                 ;
                inc     ecx                     ;
                cmp     ecx,edx                 ;
                jne     short #level_up         ;
                jmp     short #exit             ;
#level_down:                                    ;
                test    edx,edx                 ; レベルが０なら終了
                jz      short #exit             ;
#down_loop:                                     ;
                mov     ax,0202h                ; マウスレベル調整（負方向）
                call    mosmain                 ;
                dec     ecx                     ;
                cmp     cx,dx                   ;
                jne     short #down_loop        ;
#exit:                                          ;
                pop     esi                     ;
                pop     gs                      ;
                sub     eax,eax                 ;
                ret                             ;
sidework_end    endp
;-----------------------------------------------------------------------------
;       終了と割り込みハンドラの解除
;-----------------------------------------------------------------------------
                align   4
```

```
mouse_end       proc    near
                push    ds                      ;
                push    ebx                     ;
                cli                             ;
                mov     ax,0703h                ;
                call    intmain                 ; マウスイベント停止
                mov     ax,0704h                ;
                call    intmain                 ; ＳＫＢイベント停止
                sub     ebx,ebx                 ;
                mov     ah,16h                  ;
                callp   fs:[SND_OFFSET]         ; タイマＢ停止
                mov     ah,01h                  ;
                call    mosmain                 ; マウス動作終了
                test    ah,ah                   ;
                jnz     short #error            ; エラー？
                mov     ah,02h                  ;
                call    intmain                 ; ＭＯＳ割り込み処理終了
                test    eax,eax                 ;
                jnz     short #reset_extender   ; ＳＮＤハンドラ未登録？
                mov     ah,18h                  ;
                callp   fs:[SND_OFFSET]         ; タイマＢ再始動
                jmp     short #exit             ;
#reset_extender:                                ;
                mov     ax,3104h                ;
                call    sysmain                 ; read real segment
                shl     edx,16                  ;
                push    edx                     ;
                mov     ax,3103h                ;
                call    sysmain                 ; real real offset
                pop     ebx                     ;
                or      ebx,edx                 ; ebx = real offset / segment
                mov     ax,3102h                ;
                call    sysmain                 ; read native segment
                mov     ds,dx                   ;
                mov     ax,3101h                ;
                call    sysmain                 ; read native offset
                mov     cl,INT_TYPE_SOUND       ;
                mov     ax,02507h               ;
                int     21h                     ; 割り込みハンドラ解除
                sub     eax,eax                 ;
#exit:                                          ;
                pop     ebx                     ;
                pop     ds                      ;
                ret                             ;
#error:                                         ;
                sub     eax,eax                 ;
                dec     eax                     ;
                jmp     short #exit             ;
mouse_end       endp
;---------------------------------------------------------------------------
;       下請け処理
;---------------------------------------------------------------------------
                align   4
sysmain         proc    near
                callp   fs:[SYS_OFFSET]         ;
                ret                             ;
sysmain         endp
;---------------------------------------------------------------------------
                align   4
intmain         proc    near
                callp   fs:[INT_OFFSET]         ;
                ret                             ;
intmain         endp
;---------------------------------------------------------------------------
                align   4
mosmain         proc    near
                callp   fs:[MOS_OFFSET]         ;
                ret                             ;
mosmain         endp
;---------------------------------------------------------------------------
include codeend.mac
```

mos_02.asm

```
include codebgn.mac
;-----------------------------------------------------
;  マウスカーソルの表示／消去
;  MOS_disp(al)
;-----------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_disp
MOS_disp        proc    near
                test    byte ptr ds:mos_control_flg,1
                jz      short MOS_disp_exit
                mov     al,4+PAT[esp]
                $biosCall       MOS_OFFSET,02h
                movsx   eax,ah
MOS_disp_exit:
                ret
MOS_disp        endp

include codeend.mac
```

mos_03.asm

```
include codebgn.mac
;-----------------------------------------------------
;  マウスカーソル位置とボタンの読み取り
;  MOS_rdpos(ch,dx,bx)
;-----------------------------------------------------
;
                public  MOS_rdpos
MOS_rdpos       proc    near
```

```
                push    ebx
                push    edi
                $biosCall       MOS_OFFSET,03h
                mov     edi,12+PAT[esp]
                movzx   ecx,ch
                mov     [edi],ecx
                mov     edi,16+PAT[esp]
                movzx   edx,dx
                mov     [edi],edx
                mov     edi,20+PAT[esp]
                movzx   ebx,bx
                mov     [edi],ebx
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
MOS_rdpos       endp

include codeend.mac
```

mos_04.asm

```
include codebgn.mac
;-------------------------------------------------------
;  マウスカーソル位置の設定
; MOS_setpos(dx,bx)
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_setpos
MOS_setpos      proc    near
                test    byte ptr ds:mos_control_flg,2
                jz      short MOS_setpos_exit
                push    ebx
                mov     dx,8+PAT[esp]
                mov     bx,12+PAT[esp]
                $biosCall       MOS_OFFSET,04h
                movsx   eax,ah
                pop     ebx
MOS_setpos_exit:
                ret
MOS_setpos      endp

include codeend.mac
```

mos_05.asm

```
include codebgn.mac
;-------------------------------------------------------
;  ボタンの押下情報の読み取り
;  MOS_rdon(al,ch,cl,dx,bx)
;-------------------------------------------------------
;
                public  MOS_rdon
MOS_rdon        proc    near
                push    ebx
                push    edi
                mov     al,12+PAT[esp]
                $biosCall       MOS_OFFSET,05h
                mov     edi,24+PAT[esp]
                movzx   edx,dx
                mov     [edi],edx
                mov     edi,28+PAT[esp]
                movzx   ebx,bx
                mov     [edi],ebx
                mov     edi,16+PAT[esp]
                movzx   ebx,ch
                mov     [edi],ebx
                mov     edi,20+PAT[esp]
                movzx   ebx,cl
                mov     [edi],ebx
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
MOS_rdon        endp

include codeend.mac
```

mos_06.asm

```
include codebgn.mac
;-------------------------------------------------------
;  ボタンの開放情報の読み取り
; MOS_rdopen(al,ch,cl,dx,bx)
;-------------------------------------------------------
;
                public  MOS_rdopen
MOS_rdopen      proc    near
                push    ebx
                push    edi
                mov     al,12+PAT[esp]
                $biosCall       MOS_OFFSET,06h
                mov     edi,24+PAT[esp]
                movzx   edx,dx
                mov     [edi],edx
                mov     edi,28+PAT[esp]
                movzx   ebx,bx
                mov     [edi],ebx
                mov     edi,16+PAT[esp]
                movzx   ebx,ch
                mov     [edi],ebx
                mov     edi,20+PAT[esp]
```

```
                movzx   ebx,cl
                mov     [edi],ebx
                movsx   eax,ah
                pop     edi
                pop     ebx
                ret
MOS_rdopen      endp

include codeend.mac
```

mos_07.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスカーソルの水平移動範囲指定
; MOS_horizon(dx,bx)
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_horizon
MOS_horizon     proc    near
                test    byte ptr ds:mos_control_flg,4
                jz      short MOS_horizon_exit
                push    ebx
                mov     dx,8+PAT[esp]
                mov     bx,12+PAT[esp]
                $biosCall       MOS_OFFSET,07h
                movsx   eax,ah
                pop     ebx
MOS_horizon_exit:
                ret
MOS_horizon     endp

include codeend.mac
```

mos_08.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスカーソルの垂直移動範囲指定
; MOS_vertical(dx,bx)
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_vertical
MOS_vertical    proc    near
                test    byte ptr ds:mos_control_flg,8
                jz      short MOS_vertical_exit
                push    ebx
                mov     dx,8+PAT[esp]
                mov     bx,12+PAT[esp]
                $biosCall       MOS_OFFSET,08h
                movsx   eax,ah
                pop     ebx
MOS_vertical_exit:
                ret
MOS_vertical    endp

include codeend.mac
```

mos_09.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスカーソル形状の設定
; MOS_type(al,dh,dl,esi)
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_type
MOS_type        proc    near
                test    byte ptr ds:mos_control_flg,10h
                jz      short MOS_type_exit
                push    esi
                mov     al,8+PAT[esp]
                mov     dh,12+PAT[esp]
                mov     dl,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall       MOS_OFFSET,09h
                movsx   eax,ah
                pop     esi
MOS_type_exit:
                ret
MOS_type        endp

include codeend.mac
```

mos_09c.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスカーソル形状の設定（システムＲＯＭパターン展開)
;  16 色２画面合成用フルカラーマウス作成
; MOS_typeRom16(romno,x,y,buf,curcol,fuchicol)
;-------------------------------------------------------
SYSROM_SEG      equ     108h
mos_yoko        equ     32/8
mos_tate        equ     32
```

```
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_typeRom16
MOS_typeRom16   proc    near
                test    byte ptr ds:mos_control_flg,10h
                jz      #exit2
                push    ebp
                mov     ebp,esp
                push    ebx
                push    esi
                push    edi
                push    ds
                push    es
#set_segment:
                cld
                push    ds                      ; es = ds
                pop     es
                push    dword ptr SYSROM_SEG    ; es = システムＲＯＭ
                pop     ds
#set_romaddress:
                movzx   esi,byte ptr ss:[ebp+8] ; esi = アイコン番号
                test    esi,esi
                jz      short #error
                dec     esi
                cmp     esi,127
                jbe     short #main
#error:
                mov     ah,0ffh
                jmp     short #exit
#main:
                shl     esi,8
                add     esi,028000h
#set_cursolsize:
                mov     edi,ss:[ebp+20]         ; マウスドット数設定
                mov     ax,2004h
                stosw
                push    128/4                   ; ２色→16色　変換データ長
                mov     bl,ss:[ebp+24]          ; bl = カーソル色
                mov     bh,ss:[ebp+28]          ; bh = カーソルのフチの色
                shl     bl,4
                shl     ebx,4                   ; bh = 作業用データ
#setup_loop:
                mov     ecx,0408h
                sub     edx,edx
                lodsd                           ; これはサギだ!!
                not     eax
                xor     eax,ds:[esi+128-4]
#loop:
                mov     bl,bh
                shl     edx,4
                shr     al,1
                jc      short #skip01
                shr     bl,4
#skip01:
                and     bl,00001111b
                or      dl,bl
                dec     cl
                jnz     short #loop
                mov     es:[edi],edx
                add     edi,4
                shr     eax,8
                mov     cl,8
                dec     ch
                jnz     short #loop
                dec     dword ptr ss:[esp]
                jnz     short #setup_loop
                pop     ecx
#move_andpattern:
                mov     ecx,128/4
                rep     movsd
#call_tbios:
                push    es
                pop     ds
                push    dword ptr ROM_CSEG      ; fs = TBIOS
                pop     es
                mov     esi,ss:[ebp+20]
                mov     dh,ss:[ebp+12]
                mov     dl,ss:[ebp+16]
                mov     ax,0902h
                $biosCall2      MOS_OFFSET      ; マウス形状変更
#exit:
                pop     es
                pop     ds
                pop     edi
                pop     esi
                pop     ebx
                movsx   eax,ah
                leave
#exit2:
                ret

MOS_typeRom16   endp

include codeend.mac
```

mos_09r.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスカーソル形状の設定（システムＲＯＭパターン展開）
; MOS_typeRom(romno,x,y,buf)
;-------------------------------------------------------
SYSROM_SEG      equ     108h
```

```
mos_yoko        equ     32/8
mos_tate        equ     32
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_typeRom
MOS_typeRom     proc    near
                test    byte ptr ds:mos_control_flg,10h
                jz      short #exit2
                push    ebp
                mov     ebp,esp
                push    esi
                push    edi
                push    ds
                push    es
#set_segment:
                cld
                push    ds                      ; es = ds
                pop     es
                push    dword ptr SYSROM_SEG    ; es = システムＲＯＭ
                pop     ds
#set_romaddress:
                movzx   esi,byte ptr ss:[ebp+8] ; esi = アイコン番号
                test    esi,esi
                jz      short #error
                dec     esi
                cmp     esi,127
                ja      short #error            ; アイコン番号 > 127 ?
                shl     esi,8
                add     esi,028000h
#set_cursolsize:
                mov     edi,ss:[ebp+20]         ; マウスドット数設定
                mov     ax,2004h
                stosw
                mov     ecx,128/4
#loop:
                lodsd                           ; ＲＯＭパターン変換処理
                not     eax
                xor     eax,ds:[esi+128-4]
                stosd
                dec     ecx
                jnz     short #loop
#move_andpattern:
                mov     cl,128/4                ; ＡＮＤパターン転送
                rep     movsd
#call_tbios:
                push    es
                pop     ds
                push    dword ptr ROM_CSEG      ; es = TBIOS
                pop     es
                mov     esi,ss:[ebp+20]
                mov     dh,ss:[ebp+12]
                mov     dl,ss:[ebp+16]
                mov     ax,0901h
                $biosCall2      MOS_OFFSET      ; マウス形状変更
#exit:
                pop     es
                pop     ds
                pop     edi
                pop     esi
                movsx   eax,ah
                leave
#exit2:
                ret
#error:
                mov     ah,0ffh
                jmp     short #exit

MOS_typeRom     endp

include codeend.mac
```

mos_0a.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスの移動距離の読み取り
; MOS_motion(dx,bx)
;-------------------------------------------------------
;
                public  MOS_motion
MOS_motion      proc    near
                push    ebx
                push    edi
                $biosCall       MOS_OFFSET,0ah
                mov     edi,12+PAT[esp]
                movsx   edx,dx                  ;movzx -> movsx (90.03.15)
                mov     [edi],edx
                mov     edi,16+PAT[esp]
                movsx   ebx,bx
                mov     [edi],ebx
                movsx   eax,ah                  ;movzx -> movsx (90.03.15)
                pop     edi
                pop     ebx
                ret
MOS_motion      endp

include codeend.mac
```

mos_0b.asm

```
include codebgn.mac
```

```
;-----------------------------------------------------
;  ユーザ定義サブルーチンの呼び出し条件の設定
; MOS_entsub(dx,edi)
;-----------------------------------------------------
;
                public  MOS_entsub
MOS_entsub      proc    near
                push    ds
                push    esi
                mov     dx,12+PAT[esp]
                mov     esi,16+PAT[esp]
                push    cs
                pop     ds
                $biosCall       MOS_OFFSET,0bh
                movsx   eax,ah
                pop     esi
                pop     ds
                ret
MOS_entsub      endp

include codeend.mac
```

mos_0c.asm

```
include codebgn.mac

;-----------------------------------------------------
;  パルス数／画素比の設定
; MOS_pulse(dh,dl)
;-----------------------------------------------------
;
                public  MOS_pulse
MOS_pulse       proc    near
                mov     dh,4+PAT[esp]
                mov     dl,8+PAT[esp]
                $biosCall       MOS_OFFSET,0ch
                movsx   eax,ah
                ret
MOS_pulse       endp

include codeend.mac
```

mos_0d.asm

```
include codebgn.mac

;-----------------------------------------------------
;  マウス解像度の設定
; MOS_resorution(al)
;-----------------------------------------------------
;
                public  MOS_resolution
MOS_resolution  proc    near
                mov     al,4+PAT[esp]
                mov     dx,8+PAT[esp]
                $biosCall       MOS_OFFSET,0dh
                movsx   eax,ah
                ret
MOS_resolution  endp

include codeend.mac
```

mos_0da.asm

```
include codebgn.mac

;-----------------------------------------------------
;  マウスユーザラムの設定
; MOS_resolutionRam(al,cx,dx,bx,esi)
;-----------------------------------------------------
;
                public  MOS_resolutionRam
MOS_resolutionRam       proc    near
                push    ebx
                push    esi
                push    edi
                mov     al,16+PAT[esp]
                mov     cx,20+PAT[esp]
                mov     dx,24+PAT[esp]
                mov     bx,28+PAT[esp]
                mov     esi,32+PAT[esp]
                $biosCall       MOS_OFFSET,0dh
                movsx   eax,ah
                pop     edi
                pop     esi
                pop     ebx
                ret
MOS_resolutionRam       endp

include codeend.mac
```

mos_0e.asm

```
include codebgn.mac

;-----------------------------------------------------
;  マウス書き込みページの指定
; MOS_writePage(al)
;-----------------------------------------------------
;
                public  MOS_writePage
MOS_writePage   proc    near
```

```
                mov     al,4+PAT[esp]
                $biosCall       MOS_OFFSET,0eh
                movsx   eax,ah
                ret
MOS_writePage   endp

include codeend.mac
```

mos_0f.asm

```
include codebgn.mac

;--------------------------------------------------------
;  マウスカラーの設定
; MOS_color(al,edx)
;--------------------------------------------------------
;
                public  MOS_color
MOS_color       proc    near
                mov     al,4+PAT[esp]
                mov     edx,8+PAT[esp]
                $biosCall       MOS_OFFSET,0fh
                movsx   eax,ah
                ret
MOS_color       endp

include codeend.mac
```

mos_10.asm

```
include codebgn.mac

;--------------------------------------------------------
;  マウスタイルの設定
; MOS_tilePattern(bh,bl,esi)
;--------------------------------------------------------
;
                public  MOS_tilePattern
MOS_tilePattern proc    near
                push    ebx
                push    esi
                mov     bh,12+PAT[esp]
                mov     bl,16+PAT[esp]
                mov     esi,20+PAT[esp]
                $biosCall       MOS_OFFSET,010h
                movsx   eax,ah
                pop     esi
                pop     ebx
                ret
MOS_tilePattern endp
```

```
include codeend.mac
```

mos_11.asm

```
include codebgn.mac

;--------------------------------------------------------
;  マウスカーソルの水平消去範囲の設定
;  MOS_viewHorizon(al,dx,bx)
;--------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_viewHorizon
MOS_viewHorizon proc near
                test    byte ptr ds:mos_control_flg,20h
                jz      short MOS_viewHorizon_exit
                push    ebx
                mov     al,8+PAT[esp]
                mov     dx,12+PAT[esp]
                mov     bx,16+PAT[esp]
                $biosCall       MOS_OFFSET,11h
                movsx   eax,ah
                pop     ebx
MOS_viewHorizon_exit:
                ret
MOS_viewHorizon endp

include codeend.mac
```

mos_12.asm

```
include codebgn.mac

;--------------------------------------------------------
;  マウスカーソルの垂直消去範囲の設定
; MOS_viewVertical(al,dx,bx)
;--------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_viewVertical
MOS_viewVertical        proc    near
                test    byte ptr ds:mos_control_flg,40h
                jz      short MOS_viewVertical_exit
                push    ebx
                mov     al,8+PAT[esp]
                mov     dx,12+PAT[esp]
                mov     bx,16+PAT[esp]
                $biosCall       MOS_OFFSET,12h
                movsx   eax,ah
                pop     ebx
MOS_viewVertical_exit:
```

```
                ret
MOS_viewVertical        endp

include codeend.mac
```

mos_13.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       マウスボタンの左右入れ換え
;       int MOS_btnXchg(int sw);
;-----------------------------------------------------------------------------
                public  MOS_btnXchg
MOS_btnXchg     proc    near
                mov     al,ss:[esp+4]
                $biosCall       MOS_OFFSET,13h
                movsx   eax,ah
                ret
MOS_btnXchg     endp

include codeend.mac
```

mos_14.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       マウスの加速度検出機能の有効／無効の設定
;       int MOS_acceleration(int sw);
;-----------------------------------------------------------------------------
                public  MOS_acceleration
MOS_acceleration        proc    near
                mov     al,ss:[esp+4]
                $biosCall       MOS_OFFSET,14h
                movsx   eax,ah
                ret
MOS_acceleration        endp

include codeend.mac
```

mos_15.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       番号によるマウス仮想画面の設定
;       MOS_resolutionHandle(int num);
;-----------------------------------------------------------------------------
                public  MOS_resolutionHandle
MOS_resolutionHandle    proc    near
                mov     al,ss:[esp+4]
                $biosCall       MOS_OFFSET,15h
                movsx   eax,ah
                ret
MOS_resolutionHandle    endp

include codeend.mac
```

mos_gc.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスコントロールの取得
; MOS_getControl()
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOS_getControl
MOS_getControl  proc    near
                mov     eax,ds:mos_control_flg
                ret
MOS_getControl  endp

include codeend.mac
```

mos_ge.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスイベントルーチンの読みだし
; MOS_getEvent()
;-------------------------------------------------------
;
                extrn   mos_event_adr:dword
                public  MOS_getEvent
MOS_getEvent    proc    near
                pushfd
                cli
                mov     eax,ds:mos_event_adr
                popfd
                ret
MOS_getEvent    endp

include codeend.mac
```

mos_int.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスのポート読み込み
; MOS_int()
;-------------------------------------------------------
;
                public  MOS_int
MOS_int         proc    near
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[MOSINT_OFFSET]
                pop     fs
                ret
MOS_int         endp

include codeend.mac
```

mos_pe.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスポーリングインタフェースの動作終了
; MOSP_end()
;-------------------------------------------------------
;
                public  MOSP_end
MOSP_end        proc    near
                $biosCall       MOS_OFFSET,01h
                movsx   eax,ah
                ret
MOSP_end        endp

include codeend.mac
```

mos_ps.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスポーリングインタフェースの動作開始
; MOSP_start(edi,ecx)
;-------------------------------------------------------
;
                extrn   mos_control_flg:dword
                public  MOSP_start
MOSP_start      proc    near

                push    gs
                push    edi
                mov     ds:mos_control_flg,7fh
                push    ds
                pop     gs
                mov     edi,12+PAT[esp]
                mov     ecx,16+PAT[esp]
                $biosCall       MOS_OFFSET,00h
                movsx   eax,ah
                pop     edi
                pop     gs
                ret
MOSP_start      endp

include codeend.mac
```

mos_sc.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスコントロールの登録
; MOS_setControl()
;-------------------------------------------------------
                extrn   mos_control_flg:dword
                public  MOS_setControl
MOS_setControl  proc    near
                mov     eax,ss:[esp+4]
                mov     ds:mos_control_flg,eax
                xor     eax,eax
                ret
MOS_setControl  endp

include codeend.mac
```

mos_se.asm

```
include codebgn.mac

;-------------------------------------------------------
;  マウスイベントルーチンの登録
; MOS_setEvent(void *func())
;-------------------------------------------------------
MOS_eventnum    equ     003h
;-------------------------------------------------------
                extrn   mos_event_adr:dword
                public  MOS_setEvent
MOS_setEvent    proc    near
                mov     eax,ss:[esp+4]
                mov     ds:mos_event_adr,eax
                push    ds
                push    esi
```

```
                pushfd
                cli
                push    gs
                push    fs
                push    es
                push    ds
                push    cs
                push    offset user_hook
                mov     esi,esp
                push    ss
                pop     ds
                mov     al,MOS_eventnum
                $biosCall       INT_OFFSET,06h
                add     esp,24
                popfd
                pop     esi
                pop     ds
                xor     eax,eax
                ret
MOS_setEvent    endp

                align   4
user_hook       proc    far
                mov     eax,dword ptr ds:[mos_event_adr]
                test    eax,eax
                jz      short #no_event
                call    eax
#no_event:
                ret
user_hook       endp

include codeend.mac
```

mos_sev.asm

```
include codebgn.mac
;---------------------------------------------------------
;
;       マウスイベントの退避と初期化
;       void MOS_saveEvent( void *buff );
;       退避領域に３６バイトの領域が必要です。
;
;---------------------------------------------------------
saveEventStruc  struc
save_cause      dd      ?
save_offset     dd      ?
save_segment    dd      ?
save_area       dd      6 dup(?)
saveEventStruc  ends
;---------------------------------------------------------
                public  MOS_saveEvent
MOS_saveEvent   proc    near
                push    es                              ;
                push    esi                             ;
                push    dword ptr ROM_CSEG              ;
                pop     es                              ;
                mov     esi,ss:[esp+12]                 ;
                mov     ah,14h                          ; サブルーチンの読取
                call    pword ptr es:[SYS_OFFSET]       ;
                mov     ds:[esi+save_cause],eax         ;
                mov     ds:[esi+save_offset],edx        ;
                mov     ds:[esi+save_segment],ecx       ;
                lea     esi,[esi+save_area]             ;
                mov     ax,0803h                        ; マウスイベントの退避
                call    pword ptr es:[INT_OFFSET]       ;
                mov     ax,0703h                        ;
                call    pword ptr es:[INT_OFFSET]       ; マウスイベントの解除
                push    ds                              ;
                push    cs                              ;
                pop     ds                              ;
                sub     edx,edx                         ;
                mov     esi,offset #dummy_func          ;
                mov     ah,0bh                          ;
                call    pword ptr es:[MOS_OFFSET]       ; ダミールーチンの登録
                pop     ds                              ;
                pop     esi                             ;
                pop     es                              ;
                ret                                     ;
#dummy_func:                                            ;
                db      0cbh                            ;
MOS_saveEvent   endp
;---------------------------------------------------------
include codeend.mac
```

mos_rev.asm

```
include codebgn.mac
;---------------------------------------------------------
;
;       マウスイベントの復元
;       void MOS_restoreEvent( void *buff );
;
;---------------------------------------------------------
saveEventStruc  struc
save_cause      dd      ?
save_offset     dd      ?
save_segment    dd      ?
save_area       dd      6 dup(?)
saveEventStruc  ends
;---------------------------------------------------------
                        public  MOS_restoreEvent
MOS_restoreEvent        proc    near
                push    es                              ;
                push    esi                             ;
```

```
                push    dword ptr ROM_CSEG             ;
                pop     es                             ;
                mov     esi,ss:[esp+12]                ;
                mov     edx,ds:[esi+save_cause]        ;
                push    esi                            ;
                push    ds                             ;
                lds     esi,pword ptr ds:[esi+save_offset]
                mov     ah,0bh                         ;
                call    pword ptr es:[MOS_OFFSET]      ; サブルーチンの復元
                pop     ds                             ;
                pop     esi                            ;
                lea     esi,[esi+save_area]            ;
                mov     ax,0603h                       ;
                call    pword ptr es:[INT_OFFSET]      ; イベントの登録
                pop     esi                            ;
                pop     es                             ;
                ret                                    ;
MOS_restoreEvent        endp
;-----------------------------------------------------------
include codeend.mac
```

mosstk.asm

```
include databgn.mac

                public  TBIOSSTK_bottom
                db      768+768+256 dup(0)
TBIOSSTK_bottom label   byte

include dataend.mac
```

moswork.asm

```
include databgn.mac
;-------------------------------------------------------------------------------
                public  skb_event_adr
                public  mos_event_adr
                public  mos_control_flg
                public  moswork_adr
                public  moswork_seg
                public  save_mosdisp
                public  save_mosevent
                public  save_skbevent
;-------------------------------------------------------------------------------
                align   4
skb_event_adr   dd      0                       ; skb event address
mos_event_adr   dd      0                       ; call event address
mos_control_flg dd      0                       ; mouse control flag
moswork_adr     dd      0                       ; mouse address (チェック用)
moswork_seg     dw      0                       ; mouse segment (チェック用)
                dw      0
save_mosdisp    dd      0                       ; 表示状態保存フラグ
save_mosevent   dd      6 dup(?)                ; マウスイベント退避領域
save_skbevent   dd      6 dup(?)                ; ＳＫＢイベント退避領域
;-------------------------------------------------------------------------------
include dataend.mac
```

C.5 フォントBIOSサンプル

fnt_00.asm

```
include codebgn.mac

                public  FNT_ankAddr
FNT_ankAddr     proc    near
                push    ds
                push    ebx
                push    esi
                mov     al,0
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     bl,24+PAT[esp]
                $biosCall       FNT_OFFSET,00H
                mov     ebx,28+PAT[esp]
                sub     edx,edx
                mov     dx,ds
                mov     ds,8+PAT[esp]
                mov     [ebx],edx
                mov     ebx,32+PAT[esp]
                mov     [ebx],esi
                movsx   eax,ah
                pop     esi
                pop     ebx
                pop     ds
                ret
FNT_ankAddr     endp

include codeend.mac
```

fnt_00a.asm

```
include codebgn.mac
```

```
                public  FNT_ankRead
FNT_ankRead     proc    near
                push    ds
                push    ebx
                push    esi
                mov     al,1
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     bl,24+PAT[esp]
                mov     ds,28+PAT[esp]
                mov     esi,32+PAT[esp]
                $biosCall       FNT_OFFSET,00H
                movsx   eax,ah
                pop     esi
                pop     ebx
                pop     ds
                ret
FNT_ankRead     endp

include codeend.mac
```

fnt_01.asm

```
include codebgn.mac

                public  FNT_kanjiAddr
FNT_kanjiAddr   proc    near
                push    ds
                push    ebx
                push    esi
                mov     al,0
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     bx,24+PAT[esp]
                $biosCall       FNT_OFFSET,01H
                mov     ebx,28+PAT[esp]
                sub     edx,edx
                mov     dx,ds
                mov     ds,8+PAT[esp]
                mov     [ebx],edx
                mov     ebx,32+PAT[esp]
                mov     [ebx],esi
                movsx   eax,ah
                pop     esi
                pop     ebx
                pop     ds
                ret
FNT_kanjiAddr   endp

include codeend.mac
```

fnt_01a.asm

```
include codebgn.mac

                public  FNT_kanjiRead
FNT_kanjiRead   proc    near
                push    ds
                push    ebx
                push    esi
                mov     al,1
                mov     dh,16+PAT[esp]
                mov     dl,20+PAT[esp]
                mov     bx,24+PAT[esp]
                mov     ds,28+PAT[esp]
                mov     esi,32+PAT[esp]
                $biosCall       FNT_OFFSET,01H
                movsx   eax,ah
                pop     esi
                pop     ebx
                pop     ds
                ret
FNT_kanjiRead   endp

include codeend.mac
```

fnt_02.asm

```
include codebgn.mac

                public  FNT_sjisToJis
FNT_sjisToJis   proc    near
                push    ebx
                mov     bx,8+PAT[esp]
                $biosCall       FNT_OFFSET,02H
                movzx   eax,bx
                pop     ebx
                ret
FNT_sjisToJis   endp

include codeend.mac
```

fnt_03.asm

```
include codebgn.mac

                public  FNT_jisToSjis
```

```
FNT_jisToSjis   proc    near
                push    ebx
                mov     bx,8+PAT[esp]
                $biosCall       FNT_OFFSET,03H
                movzx   eax,bx
                pop     ebx
                ret
FNT_jisToSjis   endp

include codeend.mac
```

C.6 サウンドBIOSサンプル

codebgn.mac

```
page    ,132
                .386p
                assume  cs:SND_code
CGROUP          group   SND_code
SND_code        segment dword public 'CODE' use32
                include sndlib.h
```

codeend.mac

```
SND_code        ends
end
```

sndstk.asm

```
                .386p
DGROUP          group   SND_data
SND_data        segment dword public 'DATA' use32

                public  TBIOSSTK_bottom
                db      768+768+256 dup(0)
TBIOSSTK_bottom label   byte

SND_data        ends
```

```
end
```

sndlib.asm

```
page    ,132
;------------------------------------------------------------------------------
;
;       Sound bios C library for Townes
;
;------------------------------------------------------------------------------
include sndlib.h
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
;------------------------------------------------------------------------------
                .386p
                assume  cs:SND_code,ds:SND_data
DGROUP          group   SND_data
SND_data        segment dword public 'DATA' use32

                public  timer_a_int
                public  timer_b_int
                public  timer_sub_int
                public  polling_sub_int
timer_a_int     dd      ?               ;timer A INT subroutine address
timer_b_int     dd      ?               ;timer B INT subroutine address
timer_sub_int   dd      ?               ;timer sub INT subroutine address
polling_sub_int dd      ?               ;int 13 polling subroutine address

SND_data        ends
;------------------------------------------------------------------------------
;       Program Main
;------------------------------------------------------------------------------

CGROUP          group   SND_code
SND_code        segment dword public 'CODE' use32

                db      "TOWNS SOUND LIBRARY by Y.Miyazawa "
                db      "COPYRIGHT (C) 1991 FUJITSU LIMITED"

;------------------------------------------------------------------------------
;int SND_init(work)
;unsigned char *work;
;スタックに積んだＥＡＸレジスタを書き換えています
;------------------------------------------------------------------------------
SETEAX          macro   data
                mov     ss:[ebp-4],data
                endm
;------------------------------------------------------------------------------
                extrn   TBIOSSTK_bottom:byte
;------------------------------------------------------------------------------
                align   4
```

```
                public  SND_init
SND_init        proc    near

                push    ebp
                mov     ebp,esp
                pushad
                pushfd
                push    es
                push    fs
                push    gs
                push    dword ptr ROM_CSEG      ; fs = 110h
                pop     fs
#version_check:
                xor     eax,eax
                cmp     fs:[INT_OFFSET],eax     ; TBIOS-V1.1 では動作させない
                jz      short #error
#work_init:
                mov     eax,offset #dummy_return; ＥＵＰドライバ初期化処理
                mov     ds:timer_a_int,eax
                mov     ds:timer_b_int,eax
                mov     ds:timer_sub_int,eax
                mov     ds:polling_sub_int,eax
#volume_taihi:
                call    evol_read
#sound_init:
                cli
                push    ds                      ; gs = ds
                pop     gs
                mov     edi,ss:[ebp+8]
                $biosCall2      SND_OFFSET,00h  ; SOUND-BIOS 初期化
#volume_repair:
                call    evol_set
                test    al,al                   ; SOUND-BIOS 初期化正常終了？
                jz      short #init_handler
#error:
                xor     eax,eax                 ; １を返す
                inc     eax
                SETEAX  eax
                jmp     short #exit
#init_handler:
                mov     edx,offset TBIOSSTK_bottom
                $biosCall2      INT_OFFSET,03h  ; ＳＮＤハンドラ初期化
                test    eax,eax
                jz      short #timer_set        ; ＭＯＳが動作している？
#call_extender:
                mov     cl,INT_TYPE_SOUND
                mov     ax,02502h
                int     21h                     ; native mode vector read
                jc      short #error
                call    #save_native
                mov     cl,INT_TYPE_SOUND
                mov     ax,02503h
                int     21h                     ; real mode vector read
                jc      short #error
                call    #save_real
                push    ds
                mov     cl,INT_TYPE_SOUND
                lds     edx,fs:[INTERVAL]
                mov     ax,02506h               ; 割り込みベクタの登録
                int     21h
                pop     ds
                jc      short #error
#timer_set:
                mov     bl,0ffh
                mov     ecx,TIMER_A_COUNT
                mov     ah,15h                  ; timer A start
                call    sound_bios
                mov     bl,0
                mov     ah,15h                  ; timer A stop
                call    sound_bios
                mov     bl,0ffh
                mov     ecx,TIMER_B_SND
                mov     ah,16h                  ; timer B start
                call    sound_bios
                xor     eax,eax
                SETEAX  eax
#exit:
                pop     gs
                pop     fs
                pop     es
                popfd
                popad
                leave
                ret
#save_native:
                mov     al,1
                mov     edx,ebx
                call    #sysbios                ; save native offset
                mov     al,2
                xor     edx,edx
                mov     dx,es
                call    #sysbios                ; save native segment
                ret
#save_real:
                mov     al,3
                movzx   edx,bx
                call    #sysbios                ; save real offset
                mov     al,4
                shr     ebx,16
                mov     edx,ebx
                call    #sysbios                ; save real segment
                ret
#sysbios:
                mov     ah,30h
                call    pword ptr fs:[SYS_OFFSET]
#dummy_return:
```

```
                ret
SND_init        endp
;-----------------------------------------------------------------------------
;int SND_end()
;-----------------------------------------------------------------------------
                align   4
                public  SND_end
SND_end         proc    near

                push    ebx
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                pushfd
#event_clear:
                cli
                mov     al,0                    ; timerA-Event stop
                call    #intbios
                mov     al,1                    ; timerB-Event stop
                call    #intbios
                mov     al,2                    ; timerA2-Event stop
                call    #intbios
                mov     al,5                    ; polling-Event stop
                call    #intbios
#volume_taihi:
                call    evol_read
#handler_reset:
                xor     ah,ah                   ; SOUND BIOS initialize
                call    sound_bios
#volume_repair:
                call    evol_set
                xor     bh,bh
                mov     dx,2c00h
                mov     ah,11h
                call    sound_bios              ; test register off
                $biosCall2      INT_OFFSET,04h  ; ＳＮＤハンドラ解除
                test    eax,eax
                jz      short #mouse_change     ; ＭＯＳが動作している？
#sound_reset:
                push    ds
                mov     al,4
                call    #sysbios                ; read real segment
                shl     edx,16
                push    edx
                mov     al,3
                call    #sysbios                ; real real offset
                pop     ebx
                add     ebx,edx                 ; ebx = real offset / segment
                mov     al,2
                call    #sysbios                ; read native segment
                mov     ds,dx
                mov     al,1
                call    #sysbios                ; read native offset
                mov     cl,INT_TYPE_SOUND
                mov     ax,02507h               ; vector reset
                int     21h
                pop     ds
#exit:
                xor     eax,eax
                popfd
                pop     fs
                pop     ebx
                ret
#mouse_change:
                mov     bl,0ffh
                mov     ecx,TBIOS_B_MOS         ; TBIOS mouse count (23.04ms)
                mov     ah,16h
                call    sound_bios              ; timer B Reset & start
                jmp     short #exit
#sysbios:
                mov     ah,31h
                call    pword ptr fs:[SYS_OFFSET]
                ret
#intbios:
                mov     ah,07h
                call    pword ptr fs:[INT_OFFSET]
                ret

SND_end         endp
;-----------------------------------------------------------------------------
;
;       電子ボリュームの読み取り処理
;
;       in:     無し
;
;       out:    bl  = mute flag
;               ecx = Linein/CDin
;               edx = MICin/MODEMin
;
;-----------------------------------------------------------------------------
                align   4
evol_read       proc    near

                push    eax
                xor     bl,bl
                mov     ah,4ah
                call    #sndcall
                mov     ecx,edx
                call    #sndcall
                mov     ah,22h
                call    sysbios
                mov     bl,al
                pop     eax
                ret
#sndcall:
```

```
                call    sndbios
                shl     edx,16
                inc     ebx
                call    sndbios
                inc     ebx
                ret
sysbios:
                call    pword ptr fs:[SYS_OFFSET]
                ret
sndbios:
                call    pword ptr fs:[SND_OFFSET]
                ret

evol_read       endp
;------------------------------------------------------------------------------
;
;       電子ボリュームの復元処理
;
;       in:     bl  = mute flag
;               ecx = Linein/CDin
;               edx = MICin/MODEMin
;
;       out:    無し
;
;------------------------------------------------------------------------------
                align   4
evol_set        proc    near

                push    eax
                push    ebx
                mov     ah,46h
                call    sndbios
                mov     bl,3
                mov     ah,4bh
                call    #sndcall
                mov     edx,ecx
                call    #sndcall
                pop     ebx
                pop     eax
                ret
#sndcall:
                call    sndbios
                shr     edx,16
                dec     ebx
                call    sndbios
                dec     ebx
                ret

evol_set        endp
;------------------------------------------------------------------------------
SND_code        ends
end
```

sound01.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_key_on(ch,note,vol)
;int ch,note,vol;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_key_on
SND_key_on      proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]          ;channel
                mov     dh,ss:[esp+16]          ;tone
                mov     dl,ss:[esp+20]          ;volume
                mov     ah,01h                  ;key on
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_key_on      endp
include codeend.mac
```

sound02.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_key_off(ch)
;int ch;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_key_off
SND_key_off     proc    near

                push    ebx
                mov     bl,ss:[esp+8]           ;channel
                mov     ah,02h                  ;key off
                call    sound_bios
                pop     ebx
                ret

SND_key_off     endp
include codeend.mac
```

sound03.asm

```
include codebgn.mac
;-----------------------------------------------------------------------
;int SND_pan_set(ch,pan)
;int ch,pan;
;-----------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pan_set
SND_pan_set     proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]          ;channel
                mov     dl,ss:[esp+16]          ;pan
                mov     ah,03h                  ;pan set
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_pan_set     endp
include codeend.mac
```

sound04.asm

```
include codebgn.mac
;-----------------------------------------------------------------------
;int SND_inst_change(ch,inst)
;int ch,inst;
;-----------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_inst_change
SND_inst_change proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]          ;channel
                mov     dh,ss:[esp+16]          ;inst
                mov     ah,04h                  ;inst change
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_inst_change endp
include codeend.mac
```

sound05.asm

```
include codebgn.mac
;-----------------------------------------------------------------------
;int SND_inst_write(ch,inst,buff)
;int ch,inst;
;unsigned *buff;
;-----------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_inst_write
SND_inst_write  proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                mov     bl,ss:[ebp+8]           ;channel
                mov     dh,ss:[ebp+12]          ;inst number
                mov     esi,ss:[ebp+16]         ;inst data top
                mov     ah,05h                  ;inst write
                call    sound_bios
                pop     esi
                pop     edx
                pop     ebx
                leave
                ret

SND_inst_write  endp
include codeend.mac
```

sound06.asm

```
include codebgn.mac
;-----------------------------------------------------------------------
;int SND_inst_read(ch,inst,buff)
;int ch,inst;
;unsigned char *buff;
;-----------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_inst_read
SND_inst_read   proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                mov     bl,ss:[ebp+8]           ;channel
                mov     dh,ss:[ebp+12]          ;inst number
                mov     esi,ss:[ebp+16]         ;inst data top
                mov     ah,06h                  ;inst read
                call    sound_bios
                pop     esi
```

```
                pop     edx
                pop     ebx
                leave
                ret

SND_inst_read   endp
include codeend.mac
```

sound07.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pitch_change(ch,pitch)
;int ch,pitcth;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pitch_change
SND_pitch_change        proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]          ;channel
                mov     edx,ss:[esp+16]         ;picth data
                mov     ah,07h                  ;pitch change
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_pitch_change        endp
include codeend.mac
```

sound08.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_volume_change(ch,volume)
;int ch,volume;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_volume_change
SND_volume_change       proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]          ;channel
                mov     dl,ss:[esp+16]          ;volume data
                mov     ah,08h                  ;volume control
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_volume_change       endp
include codeend.mac
```

sound09.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_key_abort(ch)
;int ch;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_key_abort
SND_key_abort   proc    near

                push    ebx
                mov     bl,ss:[esp+8]           ;channel
                mov     ah,09h                  ;sound abort
                call    sound_bios
                pop     ebx
                ret

SND_key_abort   endp
include codeend.mac
```

sound0a.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int  SND_status(ch,command,*buffer)
;int  ch;
;int  command;
;char *buffer;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_status
SND_status      proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                mov     bl,ss:[ebp+8]
                mov     edx,ss:[ebp+12]
                mov     esi,ss:[ebp+16]
```

```
                mov     ah,0ah
                call    sound_bios
                pop     esi
                pop     edx
                pop     ebx
                leave
                movzx   eax,al
                ret

SND_status      endp
include codeend.mac
```

sound10.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_fm_read_status()
;-------------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_fm_read_status
SND_fm_read_status      proc    near

                push    edx
                mov     ah,10h                  ;read fm status
                call    sound_bios2
                movzx   eax,dl                  ;status
                pop     edx
                ret

SND_fm_read_status      endp
include codeend.mac
```

sound11.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_fm_write_data(bank,reg,data)
;int bank,reg,data;
;-------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_write_data
SND_fm_write_data       proc    near

                push    ebx
                push    edx
                mov     bh,ss:[esp+12]          ;bank
                mov     dh,ss:[esp+16]          ;reg
                mov     dl,ss:[esp+20]          ;data
                mov     ah,11h                  ;fm write
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_fm_write_data       endp
include codeend.mac
```

sound13.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_fm_write_save_data(bank,reg,data)
;int bank,reg,data;
;-------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_write_save_data
SND_fm_write_save_data  proc    near

                push    ebx
                push    edx
                mov     bh,ss:[esp+12]          ;bank
                mov     dh,ss:[esp+16]          ;reg
                mov     dl,ss:[esp+20]          ;data
                mov     ah,13h                  ;fm write
                call    sound_bios
                pop     edx
                pop     ebx
                ret

SND_fm_write_save_data  endp
include codeend.mac
```

sound14.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_fm_read_save_data(bank,reg,data)
;int bank,reg;
;int *data;
;-------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_read_save_data
SND_fm_read_save_data   proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
```

```
                mov     bh,[ebp+8]              ;bank
                mov     dh,[ebp+12]             ;reg
                mov     ah,14h                  ;fm read save data
                call    sound_bios
                movzx   edx,dl                  ;data
                mov     ebx,[ebp+16]
                mov     [ebx],edx
                pop     edx
                pop     ebx
                leave
                ret

SND_fm_read_save_data   endp
include codeend.mac
```

sound15.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_fm_timer_a_set(sw,count)
;int sw,count;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_timer_a_set
SND_fm_timer_a_set      proc    near

                push    ebx
                push    ecx
                mov     bl,ss:[esp+12]          ;sw
                mov     ecx,ss:[esp+16]         ;count
                mov     ah,15h                  ;fm timer a control1
                call    sound_bios
                pop     ecx
                pop     ebx
                ret

SND_fm_timer_a_set      endp
include codeend.mac
```

sound16.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_fm_timer_b_set(sw,count)
;int sw,count;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_timer_b_set
SND_fm_timer_b_set      proc    near

                push    ebx
                push    ecx
                mov     bl,ss:[esp+12]          ;sw
                mov     ecx,ss:[esp+16]         ;count
                mov     ah,16h                  ;fm timer b control1
                call    sound_bios
                pop     ecx
                pop     ebx
                ret

SND_fm_timer_b_set      endp
include codeend.mac
```

sound17.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_fm_timer_a_start()
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_timer_a_start
SND_fm_timer_a_start    proc    near

                mov     ah,17h                  ;fm timer a control2
                call    sound_bios
                ret

SND_fm_timer_a_start    endp
include codeend.mac
```

sound18.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_fm_timer_b_start()
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_timer_b_start
SND_fm_timer_b_start    proc    near

                mov     ah,18h                  ;fm timer b control2
                call    sound_bios
                ret

SND_fm_timer_b_start    endp
include codeend.mac
```

sound19.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_fm_lfo_set()
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_fm_lfo_set
SND_fm_lfo_set  proc    near

                push    edx
                mov     dl,ss:[esp+8]
                mov     ah,19h                  ;fm lfo set
                call    sound_bios
                pop     edx
                ret

SND_fm_lfo_set  endp
include codeend.mac
```

sound20.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_pcm_wave_set(buffer,wave_addr,size)
;unsigned *buffer;
;unsigned *wave_addr;
;int size;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_wave_set
SND_pcm_wave_set        proc    near

                push    ebp
                mov     ebp,esp
                push    esi
                push    ebx
                push    ecx
                mov     esi,ss:[ebp+8]          ;wave buffer address
                mov     ebx,ss:[ebp+12]         ;pcm ram address
                mov     ecx,ss:[ebp+16]         ;length
                mov     ah,20h                  ;wave memory set
                call    sound_bios
                pop     ecx
                pop     ebx
                pop     esi
                leave
                movzx   eax,al
                ret

SND_pcm_wave_set        endp
include codeend.mac
```

sound21.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_pcm_mode_set(ch)
;int ch;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_mode_set
SND_pcm_mode_set        proc    near

                push    ebx
                mov     bl,ss:[esp+8]           ;channel
                mov     ah,21h                  ;wave memory set
                call    sound_bios
                pop     ebx
                ret

SND_pcm_mode_set        endp
include codeend.mac
```

sound22.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_pcm_sound_set(work)
;unsigned char *work;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_sound_set
SND_pcm_sound_set       proc    near

                push    esi
                mov     esi,ss:[esp+8]          ;sound data top
                mov     ah,22h                  ;sound set
                call    sound_bios
                pop     esi
                ret

SND_pcm_sound_set       endp
include codeend.mac
```

sound23.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_pcm_sound_delete(id)
;int id;
;-------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_sound_delete
SND_pcm_sound_delete    proc    near

                push    edx
                mov     edx,ss:[esp+8]          ;sound ID
                mov     ah,23h                  ;sound delete
                call    sound_bios
                pop     edx
                ret

SND_pcm_sound_delete    endp
include codeend.mac
```

sound24.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_pcm_rec(freq,buff,size,trg)
;int freq;
;unsigned char *buff;
;int size;
;int trg;
;-------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_rec
SND_pcm_rec     proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    esi
                xor     bl,bl                   ;sw
                mov     ah,16h                  ;timer b stop
                call    sound_bios
                mov     edx,[ebp+8]             ;frequency
                mov     esi,[ebp+12]            ;buffer top
                mov     ecx,[ebp+16]            ;buffer size
                mov     bl,[ebp+20]             ;trigger level
                mov     ah,24h                  ;pcm rec
                call    sound_bios
                push    eax
                mov     ah,18h                  ;timer b restart
                call    sound_bios
                pop     eax
                pop     esi
                pop     edx
                pop     ecx
                pop     ebx
                leave
                ret

SND_pcm_rec     endp
include codeend.mac
```

sound24a.asm

```
include codebgn.mac
;-------------------------------------------------------------------------------
;int SND_pcm_rec2(freq,buff,size,trg)
;int freq;
;unsigned char *buff;
;int size;
;int trg;
;-------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_rec2
SND_pcm_rec2    proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    esi
                mov     edx,[ebp+8]             ;frequency
                mov     esi,[ebp+12]            ;buffer top
                mov     ecx,[ebp+16]            ;buffer size
                mov     bl,[ebp+20]             ;trigger level
                mov     ah,24h                  ;pcm rec
                call    sound_bios
                pop     esi
                pop     edx
                pop     ecx
                pop     ebx
                leave
                ret

SND_pcm_rec2    endp
include codeend.mac
```

sound24b.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_rec3(freq,buff,size,trg)
;int freq;
;unsigned char *buff;
;int size;
;int trg;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_rec3
SND_pcm_rec3    proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    esi
                mov     edx,[ebp+8]             ;frequency
                mov     esi,[ebp+12]            ;buffer top
                mov     ecx,[ebp+16]            ;buffer size
                mov     bl,[ebp+20]             ;trigger level
                mov     ah,24h                  ;pcm rec
                pushfd
                cli
                call    sound_bios
                popfd
                pop     esi
                pop     edx
                pop     ecx
                pop     ebx
                leave
                ret

SND_pcm_rec3    endp
include codeend.mac
```

sound24c.asm

```
include sndlib.h

                .386p
                assume  cs:SND_code,ds:SND_data
DGROUP          group   SND_data
SND_data        segment dword public 'DATA' use32
event_offset    dd      ?
SND_data        ends

CGROUP          group   SND_code
SND_code        segment dword public 'CODE' use32
;-----------------------------------------------------------------------------
;int SND_pcm_rec4(freq,buff,size,trg,sw,bunshu,event);
;int freq;
;unsigned char *buff;
;int size;
;int trg;
;int sw;
;int bunshu;
;void event();
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_rec4
SND_pcm_rec4    proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    esi
#mode_get:
                mov     ah,25h
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[SYS_OFFSET]
                pop     fs
                push    eax                     ;save mode
#extend_mode:
                mov     bl,1
                mov     ah,0bh
                call    sound_bios
#pcm_rec_set:
                mov     ch,[ebp+24]             ;mos button check data
                mov     cl,[ebp+28]             ; イベント分周数
                mov     edx,[ebp+32]            ; イベントのアドレス
                mov     ds:[event_offset],edx   ;
                mov     edx,offset event        ; イベント仲介ルーチン
                push    ds
                push    cs                      ;ds = cs
                pop     ds
                mov     ah,26h                  ;pcm rec set
                call    sound_bios
                pop     ds
#pcm_rec:
                mov     edx,[ebp+8]             ;frequency
                mov     esi,[ebp+12]            ;buffer top
                mov     ecx,[ebp+16]            ;buffer size
                mov     bl,[ebp+20]             ;trigger level
                mov     ah,24h                  ;pcm rec
                pushfd
                cli
                call    sound_bios
                popfd
```

```
#repair_mode:
                pop     ebx
                mov     ah,0bh
                call    sound_bios
                movzx   eax,al
                pop     esi
                pop     edx
                pop     ecx
                pop     ebx
                leave
                ret
SND_pcm_rec4    endp
                align   4
event           proc    far
                mov     eax,ds:[event_offset]
                test    eax,eax
                jz      short #exit
                call    eax
#exit:
                ret
event           endp
SND_code        ends
end
```

sound25.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_play(ch,note,volume,buffer)
;int ch;
;int note;
;int volume;
;unsigned char *buffer;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_play
SND_pcm_play    proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                mov     bl,[ebp+8]              ;channel
                mov     dh,[ebp+12]             ;note
                mov     dl,[ebp+16]             ;volume
                mov     esi,[ebp+20]            ;buffer
                mov     ah,25h                  ;pcm voice mode play
                call    sound_bios
                pop     esi
                pop     edx
                pop     ebx
                leave
                ret

SND_pcm_play    endp
include codeend.mac
```

sound25a.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_play_rom(ch,note,volume,rom_no);
;int ch;
;int note;
;int volume;
;int rom_no;
;-----------------------------------------------------------------------------
VOICE_SEG       equ     148h
VOICE_OFFSET    equ     40000h
;-----------------------------------------------------------------------------
                public  SND_pcm_play_rom
SND_pcm_play_rom        proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                push    edi
                push    ds
                push    fs
                push    gs
                mov     bl,[ebp+8]                              ;channel
                mov     dh,[ebp+12]                             ;note
                mov     dl,[ebp+16]                             ;volume
                mov     eax,[ebp+20]                            ;rom number
                push    dword ptr VOICE_SEG
                pop     ds
                push    dword ptr ROM_CSEG
                pop     fs
                lgs     edi,pword ptr fs:[SNDWORK_OFFSET]       ;work address
                mov     esi,VOICE_OFFSET
                cmp     ds:[esi],eax            ; その番号の音声は存在する？
                jb      short #error
                test    eax,eax                 ; 音声番号 = 0 ？
                jz      short #error
                mov     eax,[esi+eax*4]
                add     esi,eax                 ;voice buffer
                mov     ah,25h                  ;pcm voice mode play
                call    pword ptr fs:[SND_OFFSET]
                movzx   eax,al
```

```
#exit:
                pop     gs
                pop     fs
                pop     ds
                pop     edi
                pop     esi
                pop     edx
                pop     ebx
                leave
                ret
#error:
                xor     eax,eax                 ;-1 を返す
                dec     eax
                jmp     short #exit

SND_pcm_play_rom        endp
include codeend.mac
```

sound26.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_rec_stop()
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_rec_stop
SND_pcm_rec_stop        proc    near

                mov     ah,26h                  ;pcm voice mode play
                call    sound_bios
                ret

SND_pcm_rec_stop        endp
include codeend.mac
```

sound27.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_play_stop(ch)
;int ch;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_play_stop
SND_pcm_play_stop       proc    near

                push    ebx
                mov     bl,ss:[esp+8]           ;channel
                mov     ah,27h                  ;pcm voice mode play
                call    sound_bios
                pop     ebx
                ret

SND_pcm_play_stop       endp
include codeend.mac
```

sound28.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_status(ch)
;int ch;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_status
SND_pcm_status  proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]          ;channel
                mov     ah,28h                  ;pcm voice mode play
                call    sound_bios
                test    eax,eax
                jnz     short #skip
                movsx   eax,dl
#skip:
                pop     edx
                pop     ebx
                ret

SND_pcm_status  endp
include codeend.mac
```

sound29.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_abort()
;int ch;
;-----------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_abort
SND_pcm_abort   proc    near

                mov     ah,29h                  ;pcm abort
                call    sound_bios
                ret
```

```
SND_pcm_abort   endp
include codeend.mac
```

sound2a.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_pcm_wave_read(wave_addr,buffer,size)
;unsigned *wave_addr;
;unsigned *buffer;
;int size;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_wave_read
SND_pcm_wave_read       proc    near

                push    ebp
                mov     ebp,esp
                push    esi
                push    ebx
                push    ecx
                mov     ebx,ss:[ebp+8]          ;pcm ram address
                mov     esi,ss:[ebp+12]         ;wave buffer address
                mov     ecx,ss:[ebp+16]         ;length
                mov     ah,2ah                  ;wave memory read
                call    sound_bios
                pop     ecx
                pop     ebx
                pop     esi
                leave
                movzx   eax,al
                ret

SND_pcm_wave_read       endp
include codeend.mac
```

sound2b.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_pcm_wave_move(wave_source,wave_destination,size)
;unsigned *wave_source;
;unsigned *wave_destination;
;int size;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_wave_move
SND_pcm_wave_move       proc    near

                push    ebp
                mov     ebp,esp
                push    esi
                push    ebx
                push    ecx
                mov     esi,ss:[ebp+8]          ;wave source address
                mov     ebx,ss:[ebp+12]         ;wave destination address
                mov     ecx,ss:[ebp+16]         ;length
                mov     ah,2bh                  ;wave memory read
                call    sound_bios
                pop     ecx
                pop     ebx
                pop     esi
                leave
                movzx   eax,al
                ret

SND_pcm_wave_move       endp
include codeend.mac
```

sound2c.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_pcm_wave_write(buffer,wave_addr,size)
;unsigned *buffer;
;unsigned *wave_addr;
;int size;
;------------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_wave_write
SND_pcm_wave_write      proc    near

                push    ebp
                mov     ebp,esp
                push    esi
                push    ebx
                push    ecx
                mov     esi,[ebp+8]             ;wave buffer address
                mov     ebx,[ebp+12]            ;pcm ram address
                mov     ecx,[ebp+16]            ;length
                mov     ah,2ch
                call    sound_bios
                pop     ecx
                pop     ebx
                pop     esi
                leave
                movzx   eax,al
                ret

SND_pcm_wave_write      endp
```

```
include codeend.mac
```

sound2d.asm

```
include codebgn.mac
;---------------------------------------------------------------------------
;int SND_pcm_sound_get( int ID , char *buf );
;---------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_sound_get
SND_pcm_sound_get       proc    near
                push    esi
                mov     edx,ss:[esp+08]         ;sound ID
                mov     esi,ss:[esp+12]         ;sound data top
                mov     ah,02dh
                call    sound_bios
                pop     esi
                ret
SND_pcm_sound_get       endp
include codeend.mac
```

sound2e1.asm

```
include codebgn.mac
;---------------------------------------------------------------------------
;int SND_pcm_play2_rom(ch,note,volume,rom_no);
;int ch;
;int note;
;int volume;
;int rom_no;
;---------------------------------------------------------------------------
VOICE_SEG       equ     148h
VOICE_OFFSET    equ     40000h
;---------------------------------------------------------------------------
                public  SND_pcm_play2_rom
SND_pcm_play2_rom       proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                push    edi
                push    ds
                push    fs
                push    gs
                mov     bl,[ebp+8]                      ;channel
                mov     dh,[ebp+12]                     ;note
                mov     dl,[ebp+16]                     ;volume
                mov     eax,[ebp+20]                    ;rom number
                push    dword ptr VOICE_SEG
                pop     ds
                push    dword ptr ROM_CSEG
                pop     fs
                lgs     edi,pword ptr fs:[SNDWORK_OFFSET];work address
                mov     esi,VOICE_OFFSET
                cmp     ds:[esi],eax            ; その番号の音声は存在する？
                jb      short #error
                test    eax,eax                 ; 音声番号 = 0 ?
                jz      short #error
                mov     eax,[esi+eax*4]
                add     esi,eax                 ;voice buffer
                mov     ah,2eh                  ;pcm extand voice mode play
                call    pword ptr fs:[SND_OFFSET]
                movzx   eax,al
#exit:
                pop     gs
                pop     fs
                pop     ds
                pop     edi
                pop     esi
                pop     edx
                pop     ebx
                leave
                ret
#error:
                xor     eax,eax                 ;-1 を返す
                dec     eax
                jmp     short #exit

SND_pcm_play2_rom       endp
include codeend.mac
```

sound2e2.asm

```
include codebgn.mac
;---------------------------------------------------------------------------
;int SND_pcm_play2(ch,note,volume,buffer)
;int ch;
;int note;
;int volume;
;unsigned char *buffer;
;---------------------------------------------------------------------------
                extrn   sound_bios      :near
                public  SND_pcm_play2
SND_pcm_play2   proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
```

```
                push    edx
                push    esi
                mov     bl,[ebp+8]              ;channel
                mov     dh,[ebp+12]             ;note
                mov     dl,[ebp+16]             ;volume
                mov     esi,[ebp+20]            ;buffer
                mov     ah,2eh                  ;pcm voice mode play
                call    sound_bios
                pop     esi
                pop     edx
                pop     ebx
                leave
                ret

SND_pcm_play2   endp
include codeend.mac
```

sound40.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_joy_in_1(port,status)
;int port,*status;
;------------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_joy_in_1
SND_joy_in_1    proc    near

                push    ebp
                mov     ebp,esp
                push    edx
                mov     dh,ss:[ebp+8]
                mov     ah,40h
                call    sound_bios2
                movzx   eax,dl
                mov     edx,ss:[ebp+12]
                mov     [edx],eax
                xor     eax,eax
                pop     edx
                leave
                ret

SND_joy_in_1    endp
include codeend.mac
```

sound41.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_joy_in_2(port,status)
;int port,*status;
;------------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_joy_in_2
SND_joy_in_2    proc    near

                push    ebp
                mov     ebp,esp
                push    edx
                mov     dh,ss:[ebp+8]
                mov     ah,41h
                call    sound_bios2
                movzx   eax,dl
                mov     edx,ss:[ebp+12]
                mov     [edx],eax
                xor     eax,eax
                pop     edx
                leave
                ret

SND_joy_in_2    endp
include codeend.mac
```

sound42.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_joy_out(data,status1,status2)
;int data,*status1,*status2;
;------------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_joy_out
SND_joy_out     proc    near

                push    ebp
                mov     ebp,esp
                push    ebx
                push    edx
                push    esi
                mov     bl,ss:[ebp+8]
                mov     ah,42h
                call    sound_bios2
                mov     esi,ss:[ebp+12]
                movzx   eax,dh
                mov     [esi],eax
                mov     esi,ss:[ebp+16]
                movzx   eax,dl
                mov     [esi],eax
                xor     eax,eax
                pop     esi
```

```
                pop     edx
                pop     ebx
                leave
                ret

SND_joy_out     endp
include codeend.mac
```

sound43.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_elevol_set(num,l_vol,r_vol)
;int num,l_vol,r_vol;
;-----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_set
SND_elevol_set  proc    near

                push    ebx
                push    edx
                mov     bl,ss:[esp+12]
                mov     dh,ss:[esp+16]
                mov     dl,ss:[esp+20]
                mov     ah,43h
                call    sound_bios2
                pop     edx
                pop     ebx
                ret

SND_elevol_set  endp
include codeend.mac
```

sound44.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_elevol_init()
;-----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_init
SND_elevol_init proc    near

                mov     ah,44h
                call    sound_bios2
                ret

SND_elevol_init endp
include codeend.mac
```

sound45.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_elevol_read(num,*l_vol,*r_vol)
;int num,*l_vol,*r_vol;
;-----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_read
SND_elevol_read proc    near

                push    ebx
                push    edx
                push    esi
                mov     bl,ss:[esp+16]
                mov     ah,45h
                call    sound_bios2
                movzx   ebx,dh
                mov     esi,ss:[esp+20]
                mov     [esi],ebx
                movzx   ebx,dl
                mov     esi,ss:[esp+24]
                mov     [esi],ebx
                pop     esi
                pop     edx
                pop     ebx
                ret

SND_elevol_read endp
include codeend.mac
```

sound46.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_elevol_mute(sw)
;-----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_mute
SND_elevol_mute proc    near

                push    ebx
                mov     bl,ss:[esp+8]
                mov     ah,46h
                call    sound_bios2
                pop     ebx
                ret
```

```
SND_elevol_mute endp
include codeend.mac
```

sound47.asm

```
include codebgn.mac
;----------------------------------------------------------------------------
;int SND_elevol_led_read()
;----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_led_read
SND_elevol_led_read     proc    near

                push    edx
                mov     ah,47h
                call    sound_bios2
                movzx   eax,dl
                pop     edx
                ret

SND_elevol_led_read     endp
include codeend.mac
```

sound48.asm

```
include codebgn.mac
;----------------------------------------------------------------------------
;int SND_elevol_set(sw)
;----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_led_set
SND_elevol_led_set      proc    near

                push    edx
                mov     dl,ss:[esp+8]
                mov     ah,48h
                call    sound_bios2
                pop     edx
                ret

SND_elevol_led_set      endp
include codeend.mac
```

sound49.asm

```
include codebgn.mac
;----------------------------------------------------------------------------
;int SND_elevol_all_mute(sw)
;----------------------------------------------------------------------------
                extrn   sound_bios2     :near
                public  SND_elevol_all_mute
SND_elevol_all_mute     proc    near

                push    edx
                mov     dl,ss:[esp+8]
                mov     ah,49h
                call    sound_bios2
                pop     edx
                ret

SND_elevol_all_mute     endp
include codeend.mac
```

sounden1.asm

```
include codebgn.mac
;----------------------------------------------------------------------------
;       sound bios entry point
;----------------------------------------------------------------------------
                public  SND_driver
                public  sound_bios
sound_bios      proc    near
SND_driver:
                push    fs
                push    gs
                push    edi
                push    dword ptr ROM_CSEG
                pop     fs
                lgs     edi,pword ptr fs:[SNDWORK_OFFSET]
                call    pword ptr fs:[SND_OFFSET]
                movzx   eax,al
                pop     edi
                pop     gs
                pop     fs
                ret

sound_bios      endp
include codeend.mac
```

sounden2.asm

```
include codebgn.mac
;----------------------------------------------------------------------------
;       sound bios entry point
;----------------------------------------------------------------------------
                public  sound_bios2
```

```
sound_bios2     proc    near

                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[SND_OFFSET]
                movzx   eax,al
                pop     fs
                ret

sound_bios2     endp
include codeend.mac
```

snd_sev.asm

```
include codebgn.mac
;-----------------------------------------------------------------------
;
;       void SND_saveEvent( void *buff );
;
;       サウンドイベントの退避と初期化
;       ９６バイトの退避領域を必要とします。
;
;-----------------------------------------------------------------------
                public  SND_saveEvent
SND_saveEvent   proc    near
                push    es                              ;
                push    ebx                             ;
                push    esi                             ;
                push    edi                             ;
                push    dword ptr ROM_CSEG              ;
                pop     es                              ;
                mov     edi,ss:[esp+20]                 ;
                sub     ebx,ebx                         ;
#loop:                                                  ;
                mov     esi,edi                         ;
                mov     al,bl                           ;
                mov     ah,08h                          ;
                call    pword ptr es:[INT_OFFSET]       ; イベントの取得
                mov     al,bl                           ;
                mov     ah,07h                          ;
                call    pword ptr es:[INT_OFFSET]       ; イベントの解除
                add     edi,24                          ;
                inc     ebx                             ;
                cmp     ebx,2                           ;
                jbe     short #loop                     ;
                mov     esi,edi                         ;
                mov     ax,0805h                        ;
                call    pword ptr es:[INT_OFFSET]       ; イベントの取得
                mov     ax,0705h                        ;
                call    pword ptr es:[INT_OFFSET]       ; イベントの解除
                pop     edi                             ;
                pop     esi                             ;
                pop     ebx                             ;
                pop     es                              ;
                ret                                     ;
SND_saveEvent   endp
include codeend.mac
```

snd_rev.asm

```
include codebgn.mac
;-----------------------------------------------------------------------
;
;       void SND_restoreEvent( void *buff );
;
;       サウンドイベントの復元
;
;-----------------------------------------------------------------------
                public  SND_restoreEvent
SND_restoreEvent        proc    near
                push    es                              ;
                push    ebx                             ;
                push    esi                             ;
                push    edi                             ;
                push    dword ptr ROM_CSEG              ;
                pop     es                              ;
                mov     edi,ss:[esp+20]                 ;
                sub     ebx,ebx                         ;
#loop:                                                  ;
                mov     esi,edi                         ;
                mov     al,bl                           ;
                mov     ah,06h                          ;
                call    pword ptr es:[INT_OFFSET]       ; イベントの登録
                add     edi,24                          ;
                inc     ebx                             ;
                cmp     ebx,2                           ;
                jbe     short #loop                     ;
                mov     esi,edi                         ;
                mov     ax,0605h                        ;
                call    pword ptr es:[INT_OFFSET]       ; イベントの登録
                pop     edi                             ;
                pop     esi                             ;
                pop     ebx                             ;
                pop     es                              ;
                ret                                     ;
SND_restoreEvent        endp
include codeend.mac
```

sndevol.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_get_elevol_set(int num,int *l_vol,int *r_vol)
;-----------------------------------------------------------------------------
                public  SND_get_elevol_set
SND_get_elevol_set      proc    near

                mov     al,ss:[esp+4]
                $biosCall       SYS_OFFSET,021h
                test    eax,eax                 ; 符号有りならエラー
                jg      short #exit
#set_leftvolume:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #set_rightvolume
                movzx   ecx,ah
                mov     ds:[edx],ecx
#set_rightvolume:
                mov     edx,ss:[esp+12]
                test    edx,edx
                jz      short #exit
                movzx   ecx,al
                mov     ds:[edx],ecx
#exit:
                ret

SND_get_elevol_set      endp
include codeend.mac
```

sndevolm.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_get_elevol_mute(int *sw);
;-----------------------------------------------------------------------------
                public  SND_get_elevol_mute
SND_get_elevol_mute     proc    near

                $biosCall       SYS_OFFSET,022h
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #exit
                mov     ds:[edx],eax
#exit:
                ret

SND_get_elevol_mute     endp
include codeend.mac
```

sndpig.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_sound_ID_get( char *buf );
;-----------------------------------------------------------------------------
                public  SND_pcm_sound_ID_get
SND_pcm_sound_ID_get    proc    near
                mov     edx,ss:[esp+04]         ;sound ID
                mov     ah,023h
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[SYS_OFFSET]
                pop     fs
                ret
SND_pcm_sound_ID_get    endp
include codeend.mac
```

sndpmg.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_pcm_mode_get( void );
;-----------------------------------------------------------------------------
                public  SND_pcm_mode_get
SND_pcm_mode_get        proc    near
                mov     ah,024h
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                call    pword ptr fs:[SYS_OFFSET]
                pop     fs
                ret
SND_pcm_mode_get        endp
include codeend.mac
```

soundtas.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_int_timer_a_set(addr)
;unsigned char *addr;
;-----------------------------------------------------------------------------
SND_timerAnum   equ     0
;-----------------------------------------------------------------------------
```

```
                extrn   timer_a_int     :dword
                public  SND_int_timer_a_set
SND_int_timer_a_set     proc    near
                mov     eax,[esp+4]             ;address
                mov     ds:timer_a_int,eax
                push    ds
                push    esi
                pushfd
                cli
                push    gs
                push    fs
                push    es
                push    ds
                push    cs
                push    offset user_hook
                mov     esi,esp
                push    ss
                pop     ds
                mov     al,SND_timerAnum
                $biosCall       INT_OFFSET,06h
                add     esp,24
                popfd
                pop     esi
                pop     ds
                xor     eax,eax
                ret
SND_int_timer_a_set     endp

                align   4
user_hook       proc    far
                call    dword ptr ds:[timer_a_int]
                ret
user_hook       endp

include codeend.mac
```

soundtbs.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;int SND_int_timer_b_set(addr)
;unsigned char *addr;
;-----------------------------------------------------------------------------
SND_timerBnum   equ     1
;-----------------------------------------------------------------------------
                extrn   timer_b_int     :dword
                public  SND_int_timer_b_set
SND_int_timer_b_set     proc    near
                mov     eax,[esp+4]             ;address
                mov     ds:timer_b_int,eax
                push    ds
                push    esi
                pushfd
                cli
                push    gs
                push    fs
                push    es
                push    ds
                push    cs
                push    offset user_hook
                mov     esi,esp
                push    ss
                pop     ds
                mov     al,SND_timerBnum
                $biosCall       INT_OFFSET,06h
                add     esp,24
                popfd
                pop     esi
                pop     ds
                xor     eax,eax
                ret
SND_int_timer_b_set     endp

                align   4
user_hook       proc    far
                call    dword ptr ds:[timer_b_int]
                ret
user_hook       endp

include codeend.mac
```

soundtag.asm

```
include codebgn.mac
;-----------------------------------------------------------------------------
;char *SND_int_timer_a_get()
;-----------------------------------------------------------------------------
                extrn   timer_a_int     :dword
                public  SND_int_timer_a_get
SND_int_timer_a_get     proc    near

                mov     eax,ds:timer_a_int
                ret

SND_int_timer_a_get     endp
include codeend.mac
```

soundtbg.asm

```
include codebgn.mac
```

```
;------------------------------------------------------------------------
;char *SND_int_timer_b_get()
;------------------------------------------------------------------------
                extrn   timer_b_int     :dword
                public  SND_int_timer_b_get
SND_int_timer_b_get     proc    near

                mov     eax,ds:timer_b_int
                ret

SND_int_timer_b_get     endp
include codeend.mac
```

soundtss.asm

```
include codebgn.mac
;------------------------------------------------------------------------
;int SND_int_timer_sub_set(addr)
;char *addr;
;------------------------------------------------------------------------
SND_timerA2num  equ     2
;------------------------------------------------------------------------
                extrn   timer_sub_int   :dword
                public  SND_int_timer_sub_set
SND_int_timer_sub_set   proc    near
                mov     eax,[esp+4]                 ;address
                mov     ds:timer_sub_int,eax
                push    ds
                push    esi
                pushfd
                cli
                push    gs
                push    fs
                push    es
                push    ds
                push    cs
                push    offset user_hook
                mov     esi,esp
                push    ss
                pop     ds
                mov     al,SND_timerA2num
                $biosCall       INT_OFFSET,06h
                add     esp,24
                popfd
                pop     esi
                pop     ds
                xor     eax,eax
                ret
SND_int_timer_sub_set   endp

                align   4

user_hook       proc    far
                call    dword ptr ds:[timer_sub_int]
                ret
user_hook       endp

include codeend.mac
```

soundtsg.asm

```
include codebgn.mac
;------------------------------------------------------------------------
;char *SND_int_timer_sub_get()
;------------------------------------------------------------------------
                extrn   timer_sub_int   :dword
                public  SND_int_timer_sub_get
SND_int_timer_sub_get   proc    near

                mov     eax,ds:timer_sub_int
                ret

SND_int_timer_sub_get   endp
include codeend.mac
```

soundpss.asm

```
include codebgn.mac
;------------------------------------------------------------------------
;int SND_int_polling_sub_set(addr)
;char *addr;
;------------------------------------------------------------------------
KAKUSHI_num     equ     5
;------------------------------------------------------------------------
                extrn   polling_sub_int :dword
                public  SND_int_polling_sub_set
SND_int_polling_sub_set proc    near
                mov     eax,[esp+4]                 ;address
                mov     ds:polling_sub_int,eax
                push    ds
                push    esi
                pushfd
                cli
                push    gs
                push    fs
                push    es
                push    ds
                push    cs
                push    offset user_hook
                mov     esi,esp
                push    ss
```

```
                pop     ds
                mov     al,KAKUSHI_num
                $biosCall       INT_OFFSET,06h
                add     esp,24
                popfd
                pop     esi
                pop     ds
                xor     eax,eax
                ret
SND_int_polling_sub_set endp

                align   4
user_hook       proc    far
                call    dword ptr ds:[polling_sub_int]
                ret
user_hook       endp

include codeend.mac
```

soundpsg.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;char *SND_int_polling_sub_get()
;------------------------------------------------------------------------------
                extrn   polling_sub_int :dword
                public  SND_int_polling_sub_get
SND_int_polling_sub_get proc    near

                mov     eax,ds:polling_sub_int
                ret

SND_int_polling_sub_get endp
include codeend.mac
```

soundgis.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_get_int_status(void);
;------------------------------------------------------------------------------
                public  SND_get_int_status
SND_get_int_status      proc    near
                push    fs
                push    dword ptr ROM_CSEG
                pop     fs
                mov     ah,009h
                call    pword ptr fs:[INT_OFFSET]
                pop     fs
                ret
SND_get_int_status      endp
include codeend.mac
```

soundgbs.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_get_boot_status(void);
;------------------------------------------------------------------------------
                extrn   TOS_cmosRead    :near
                public  SND_get_boot_status
SND_get_boot_status     proc    near
                call    TOS_cmosRead
                push    ds
                push    60h
                pop     ds
                sub     ecx,ecx
                cmp     ds:[eax-2],cx
                je      short #error
                lea     edx,[eax+148h]
                mov     eax,ds:[edx]
                mov     ds:[edx],cl
#exit:
                pop     ds
                ret
#error:
                sub     eax,eax
                dec     eax
                jmp     short #exit
SND_get_boot_status     endp
include codeend.mac
```

soundiss.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_int_stack_set( void *para );
;------------------------------------------------------------------------------
                public  SND_int_stack_set
SND_int_stack_set       proc    near
                push    esi
                mov     esi,ss:[esp+8]
                sub     edx,edx
                $biosCall       INT_OFFSET,0bh
                pop     esi
                ret
SND_int_stack_set       endp
include codeend.mac
```

soundisg.asm

```
include codebgn.mac
;------------------------------------------------------------------------------
;int SND_int_stack_get( void *para );
;------------------------------------------------------------------------------
                public  SND_int_stack_get
SND_int_stack_get       proc    near
                push    esi
                mov     esi,ss:[esp+8]
                sub     edx,edx
                $biosCall       INT_OFFSET,0ch
                pop     esi
                ret
SND_int_stack_get       endp
include codeend.mac
```

C.7　システム情報BIOSサンプル

sys_01.asm

```
include codebgn.mac

                        public  EGB_getResolution
EGB_getResolution       proc    near
                $biosCall       SYS_OFFSET,01h
#checknull_page0:
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #checknull_page1
                movzx   ecx,al
                mov     ds:[edx],ecx
#checknull_page1:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                movzx   ecx,ah
                mov     ds:[edx],ecx
#exit:
                ret
EGB_getResolution       endp

include codeend.mac
```

sys_02.asm

```
include codebgn.mac

                        public  EGB_getWritePage
EGB_getWritePage        proc    near
                mov     edx,ss:[esp+4]          ; edx = offset
                mov     ecx,ss:[esp+8]          ; ecx = segment
                $biosCall       SYS_OFFSET,02h
                ret
EGB_getWritePage        endp

include codeend.mac
```

sys_03.asm

```
include codebgn.mac

                        public  EGB_getDisplayPage
EGB_getDisplayPage      proc    near
                $biosCall       SYS_OFFSET,03h
#checknull_pri:
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #checknull_disp
                movzx   ecx,al
                mov     ds:[edx],ecx
#checknull_disp:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                movzx   ecx,ah
                mov     ds:[edx],ecx
#exit:
                ret
EGB_getDisplayPage      endp

include codeend.mac
```

sys_04.asm

```
include codebgn.mac
```

```
                        public  EGB_getDisplayStart
EGB_getDisplayStart     proc    near
                mov     cl,ss:[esp+4]
                mov     ch,ss:[esp+8]
                $biosCall       SYS_OFFSET,04h
#checknull_x:
                mov     edx,ss:[esp+12]
                test    edx,edx
                jz      short #checknull_y
                movzx   ecx,ax
                mov     ds:[edx],ecx
#checknull_y:
                mov     edx,ss:[esp+16]
                test    edx,edx
                jz      short #exit
                mov     ecx,eax
                shr     ecx,16
                mov     ds:[edx],ecx
#exit:
                ret
EGB_getDisplayStart     endp

include codeend.mac
```

sys_05.asm

```
include codebgn.mac

                        public  EGB_getPalette
EGB_getPalette          proc    near
                mov     al,ss:[esp+4]
                mov     edx,ss:[esp+8]
                $biosCall       SYS_OFFSET,05h
                ret
EGB_getPalette          endp

include codeend.mac
```

sys_06.asm

```
include codebgn.mac

                        public  EGB_getGa3
EGB_getGa3              proc    near
                mov     edx,ss:[esp+4]
                $biosCall       SYS_OFFSET,06h
                ret
EGB_getGa3              endp

include codeend.mac
```

sys_07.asm

```
include codebgn.mac

                        public  EGB_getGa4
EGB_getGa4              proc    near
                mov     edx,ss:[esp+4]
                $biosCall       SYS_OFFSET,07h
                ret
EGB_getGa4              endp

include codeend.mac
```

sys_08.asm

```
include codebgn.mac

                        public  EGB_setGa3
EGB_setGa3              proc    near
                mov     edx,ss:[esp+4]
                $biosCall       SYS_OFFSET,08h
                ret
EGB_setGa3              endp

include codeend.mac
```

sys_09.asm

```
include codebgn.mac

                        public  EGB_setGa4
EGB_setGa4              proc    near
                mov     edx,ss:[esp+4]
                $biosCall       SYS_OFFSET,09h
                ret
EGB_setGa4              endp

include codeend.mac
```

sys_0a.asm

```
include codebgn.mac
```

```
                public  EGB_getModeInfo
EGB_getModeInfo proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    esi
                mov     al,ss:[ebp+8]
                $biosCall       SYS_OFFSET,0Ah
                test    eax,eax
                jnz     short #exit
#set_color:
                mov     esi,ss:[ebp+28]
                test    esi,esi
                jz      short #set_vx
                mov     ds:[esi],ecx
#set_vx:
                mov     esi,ss:[ebp+12]
                test    esi,esi
                jz      short #set_vy
                movzx   ecx,dx
                mov     ds:[esi],ecx
#set_vy:
                mov     esi,ss:[ebp+16]
                test    esi,esi
                jz      short #set_dx
                mov     ecx,edx
                shr     ecx,16
                mov     ds:[esi],ecx
#set_dx:
                mov     esi,ss:[ebp+20]
                test    esi,esi
                jz      short #set_dy
                movzx   ecx,bx
                mov     ds:[esi],ecx
#set_dy:
                mov     esi,ss:[ebp+24]
                test    esi,esi
                jz      short #exit
                mov     ecx,ebx
                shr     ecx,16
                mov     ds:[esi],ecx
#exit:
                pop     esi
                pop     ebx
                leave
                ret
EGB_getModeInfo endp

include codeend.mac
```

sys_0b.asm

```
include codebgn.mac

                public  EGB_getDisplayInfo
EGB_getDisplayInfo      proc    near
                mov     al,ss:[esp+04]
                mov     ecx,ss:[esp+08]
                $biosCall       SYS_OFFSET,0Bh
                ret
EGB_getDisplayInfo      endp

include codeend.mac
```

sys_11.asm

```
include codebgn.mac

;--------------------------------------------------------
;       マウス表示／消去の状態読み取り
;       int MOS_getDisp(int *n,int *level);
;--------------------------------------------------------
;
                public  MOS_getDisp
MOS_getDisp     proc    near
                $biosCall       SYS_OFFSET,11h
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #set_level
                movzx   ecx,al
                mov     ds:[edx],ecx
#set_level:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                mov     ecx,eax
                shr     ecx,16
                mov     ds:[edx],ecx
#exit:
                ret
MOS_getDisp     endp

include codeend.mac
```

sys_12.asm

```
include codebgn.mac
```

```
;------------------------------------------------------------------------------
;       マウスの水平移動範囲の読み取り
;       int MOS_getHorizon(int *xmin,int *xmax);
;------------------------------------------------------------------------------
                public  MOS_getHorizon
MOS_getHorizon  proc    near
                $biosCall       SYS_OFFSET,12h
#set_min:
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #set_max
                movzx   ecx,ax
                mov     ds:[edx],ecx
#set_max:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                mov     ecx,eax
                shr     ecx,16
                mov     ds:[edx],ecx
#exit:
                ret
MOS_getHorizon  endp

include codeend.mac
```

sys_13.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       マウスの垂直移動範囲の読み取り
;       int MOS_getVertical(int *ymin,int *ymax);
;------------------------------------------------------------------------------
                public  MOS_getVertical
MOS_getVertical proc    near
                $biosCall       SYS_OFFSET,13h
#set_min:
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #set_max
                movzx   ecx,ax
                mov     ds:[edx],ecx
#set_max:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                mov     ecx,eax
                shr     ecx,16
                mov     ds:[edx],ecx
#exit:
                ret
MOS_getVertical endp

include codeend.mac
```

sys_14.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       サブルーチンの登録状態の読み取り
;       int MOS_getEntsub(int *term,void (**sub_offset)(),int *sub_seg);
;------------------------------------------------------------------------------
                public  MOS_getEntsub
MOS_getEntsub   proc    near
                push    ebx
                $biosCall       SYS_OFFSET,14h
#set_term:
                mov     ebx,ss:[esp+8]
                mov     ds:[ebx],eax
                mov     ebx,ss:[esp+12]
                mov     ds:[ebx],edx
                mov     ebx,ss:[esp+16]
                mov     ds:[ebx],ecx
                pop     ebx
                ret
MOS_getEntsub   endp

include codeend.mac
```

sys_15.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       パルス数/画素比の設定値の読み取り
;       int MOS_getPulse(int *x,int *y);
;------------------------------------------------------------------------------
                public  MOS_getPulse
MOS_getPulse    proc    near
                $biosCall       SYS_OFFSET,15h
#set_x:
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #set_y
                movzx   ecx,ax
                mov     ds:[edx],ecx
#set_y:
                mov     edx,ss:[esp+8]
```

```
                test    edx,edx
                jz      short #exit
                mov     ecx,eax
                shr     ecx,16
                mov     ds:[edx],ecx
#exit:
                ret
MOS_getPulse    endp

include codeend.mac
```

sys_16.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       仮想画面の設定状態の読み取り
;       int MOS_getResolution(int *page0,int *page1);
;-----------------------------------------------------------------------------
                        public  MOS_getResolution
MOS_getResolution       proc    near
                $biosCall       SYS_OFFSET,16h
#set_page0:
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #set_page1
                movzx   ecx,al
                mov     ds:[edx],ecx
#set_page1:
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                movzx   ecx,ah
                mov     ds:[edx],ecx
#exit:
                ret
MOS_getResolution       endp

include codeend.mac
```

sys_17.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       書き込みページの読み取り
;       int MOS_getWritePage(void);
;-----------------------------------------------------------------------------
                        public  MOS_getWritePage
MOS_getWritePage        proc    near
                $biosCall       SYS_OFFSET,17h
                ret
MOS_getWritePage        endp

include codeend.mac
```

sys_18.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       マウスボタンの入れ換え状態の取得
;       int MOS_getBtnXchg(int *sw);
;-----------------------------------------------------------------------------
                public  MOS_getBtnXchg
MOS_getBtnXchg  proc    near

                $biosCall       SYS_OFFSET,18h
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #exit
                mov     ds:[edx],eax
#exit:
                ret

MOS_getBtnXchg  endp

include codeend.mac
```

sys_19.asm

```
include codebgn.mac

;-----------------------------------------------------------------------------
;       マウスの加速度検出機能の動作状態の取得
;       int MOS_getAcceleration(int *sw);
;-----------------------------------------------------------------------------
                        public  MOS_getAcceleration
MOS_getAcceleration     proc    near
                $biosCall       SYS_OFFSET,19h
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #exit
                mov     ds:[edx],eax
#exit:
                ret
MOS_getAcceleration     endp
```

```
include codeend.mac
```

sys_1a.asm

```
include codebgn.mac

;--------------------------------------------------------
;  マウス動作状態の取得
; MOS_openCheck()
;--------------------------------------------------------
                public  MOS_openCheck
MOS_openCheck   proc    near

                $biosCall       SYS_OFFSET,1ah
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #exit
                mov     ds:[edx],eax
#exit:
                ret
MOS_openCheck   endp

include codeend.mac
```

sys_30.asm

```
include codebgn.mac

;---------------------------------------------------------------------------
;       マウスの動作検出時間の設定
;       int MOS_checkTime(int time);
;---------------------------------------------------------------------------
                public  MOS_checkTime
MOS_checkTime   proc    near
                mov     edx,ss:[esp+4]
                mov     al,5
                $biosCall       SYS_OFFSET,30h
                ret
MOS_checkTime   endp

include codeend.mac
```

sys_31.asm

```
include codebgn.mac

;---------------------------------------------------------------------------
;       マウスダブルクリック時間の読み取り
;       int MOS_getCheckTime(int *time);
;---------------------------------------------------------------------------
                public  MOS_getCheckTime
MOS_getCheckTime        proc    near

                mov     al,5
                $biosCall       SYS_OFFSET,31h
                mov     eax,edx
                mov     edx,ss:[esp+4]
                test    edx,edx
                jz      short #exit
                mov     ds:[edx],eax
#exit:
                ret

MOS_getCheckTime        endp

include codeend.mac
```

sys_40.asm

```
include codebgn.mac

;---------------------------------------------------------------------------
;       パラメータでの解像度の取得
;       EGB_getResolutionIndex( int video , int totalPage , void *para );
;---------------------------------------------------------------------------
                        public  EGB_getResolutionIndex
EGB_getResolutionIndex  proc    near
                mov     al,ss:[esp+4]
                mov     cl,ss:[esp+8]
                mov     edx,ss:[esp+12]
                $biosCall       SYS_OFFSET,40h
                ret
EGB_getResolutionIndex  endp

include codeend.mac
```

sys_41.asm

```
include codebgn.mac

;---------------------------------------------------------------------------
;       指定したページの解像度の取得
;       EGB_getResolutionPage( int page , char *para );
;---------------------------------------------------------------------------
                        public  EGB_getResolutionPage
```

```
EGB_getResolutionPage   proc    near
                mov     al,ss:[esp+4]
                mov     ecx,ss:[esp+8]
                $biosCall       SYS_OFFSET,41h
                ret
EGB_getResolutionPage   endp

include codeend.mac
```

sys_42.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       指定したピクセル数（色数）での解像度の取得
;       EGB_getResolutionMax( int video , void *color , void *buf );
;------------------------------------------------------------------------------
                        public  EGB_getResolutionMax
EGB_getResolutionMax    proc    near
                mov     al,ss:[esp+4]
                mov     edx,ss:[esp+8]
                mov     ecx,ss:[esp+12]
                $biosCall       SYS_OFFSET,42h
                ret
EGB_getResolutionMax    endp

include codeend.mac
```

sys_43.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       画面モード番号での解像度の取得
;       EGB_getResolutionMode( void *mode , void *buf );
;------------------------------------------------------------------------------
                        public  EGB_getResolutionMode
EGB_getResolutionMode   proc    near
                mov     edx,ss:[esp+4]
                mov     ecx,ss:[esp+8]
                $biosCall       SYS_OFFSET,43h
                ret
EGB_getResolutionMode   endp

include codeend.mac
```

sys_44.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       現在表示可能なページの取得
;       EGB_getActivePage( int handle, void *display );
;------------------------------------------------------------------------------
                        public  EGB_getActivePage
EGB_getActivePage       proc    near
                mov     dx,ss:[esp+4]
                $biosCall       SYS_OFFSET,44h
                mov     edx,ss:[esp+8]
                test    edx,edx
                jz      short #exit
                mov     ds:[edx],eax
#exit:
                ret
EGB_getActivePage       endp

include codeend.mac
```

sys_45.asm

```
include codebgn.mac

;------------------------------------------------------------------------------
;       パレット有効ビットの取得
;       EGB_getActivePalette( int handle , void *palette );
;------------------------------------------------------------------------------
                        public  EGB_getActivePalette
EGB_getActivePalette    proc    near
                mov     dx,ss:[esp+4]
                mov     ecx,ss:[esp+8]
                $biosCall       SYS_OFFSET,45h
                ret
EGB_getActivePalette    endp

include codeend.mac
```

sys_46.asm

```
include codebgn.mac

                public  EGB_getActiveBit
EGB_getActiveBit        proc    near
                mov     dx,ss:[esp+04]
                mov     ecx,ss:[esp+08]
```

```
                $biosCall       SYS_OFFSET,46h
                ret
EGB_getActiveBit        endp

include codeend.mac
```

C.8 拡張サウンドBIOSサンプル

wavlib.h

```
.xlist
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Program Name : WAV.LIB
;*  File    Name : WAVLIB.H
;*  Function     : Wave BIOS C ライブラリ 定義情報
;*
;*********************************************************************

;*********************************************************************
;* ROM のセレクタ値定義
;*********************************************************************
ROM_SEG         EQU     0110h
ROM_DSEG        EQU     0118h

;*********************************************************************
;* WAVE BIOS のオフセット値 (SOUND BIOS と同じ値)
;*********************************************************************
WAV_OFFSET              equ     0080h

;*********************************************************************
;* ライブラリの入力値の取り出し
;*********************************************************************
PRM1            EQU     ss:[ebp+08]
PRM2            EQU     ss:[ebp+12]
PRM3            EQU     ss:[ebp+16]
PRM4            EQU     ss:[ebp+20]
PRM5            EQU     ss:[ebp+24]
PRM6            EQU     ss:[ebp+28]
PRM7            EQU     ss:[ebp+32]
PRM8            EQU     ss:[ebp+36]
PRM9            EQU     ss:[ebp+40]

;*********************************************************************
;* 録音/再生前準備機能のパラメタの構造体
;*********************************************************************
PREPAREPRM      STRUC
dwDataOffset    DD      0               ; 録音/再生データオフセット
wDataSelector   DW      0               ; 録音/再生データセレクタ
                DW      0               ; align 用折舒
dwFuncOffset    DD      0               ; 処理関数オフセット
wFuncSelector   DW      0               ; 処理関数セレクタ
                DW      0               ; align 用折舒
dwStackOffset   DD      0               ; ローカルスタックオフセット
wStackSelector  DW      0               ; ローカルスタックセレクタ
                DW      0               ; align 用折舒
wFuncDS         DW      0               ; 処理関数実行時に設定する DS
                DW      0               ; align 用折舒
wFuncES         DW      0               ; 処理関数実行時に設定する ES
                DW      0               ; align 用折舒
wFuncFS         DW      0               ; 処理関数実行時に設定する FS
                DW      0               ; align 用折舒
wFuncGS         DW      0               ; 処理関数実行時に設定する GS
                DW      0               ; align 用折舒
PREPAREPRM      ENDS

;*********************************************************************
;* リングバッファ関連テーブルアドレス退避領域の構造体
;*********************************************************************
PUSHTABLE       STRUC
dwRingOff       DD      0               ; リングバッファアドレスオフセット
wRingSel        DW      0               ; リングバッファアドレスセレクタ
                DW      0               ; align 用折舒
dwRingCtrlOff   DD      0               ; リングバッファ管理テーブルオフセット
wRingCtrlSel    DW      0               ; リングバッファ管理テーブルセレクタ
                DW      0               ; align 用折舒
PUSHTABLE       ENDS

.list
```

codebgn.mac

```
page    ,132
include ..\inc\wavlib.h
include ..\inc\wavlib.mac
                .386p
                assume  cs:WAV_code,ds:WAV_data
DGROUP          group   WAV_data
WAV_data        segment dword public 'DATA' use32
WAV_data        ends
CGROUP          group   WAV_code
```

```
WAV_code        segment dword public 'CODE' use32
```

codeend.mac

```
WAV_code        ends
                end
```

wavlib.mac

```
.xlist
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Program Name : WAV.LIB
;*  File    Name : WAVLIB.MAC
;*  Function     : Wave BIOS C ライブラリ マクロ定義
;*
;*********************************************************************

;*********************************************************************
;* BIOS 呼び出しマクロ
;*********************************************************************
$biosCall       macro   rom,data
                mov     ah,data
                push    fs
                push    dword ptr ROM_SEG
                pop     fs
                call    pword ptr fs:[rom]
                pop     fs
                endm
.list
```

wav60.asm

```
include ..\inc\codebgn.mac
                extrn   WAV_DummyFunc:near

                extrn   dwUserFunc:dword
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_init( ) ;
;*  Function    : ライブラリの初期化
;*
;*                 [入力]
;*                   なし
;*
;*                 [戻値]
;*                   =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_init
WAV_init        proc    near
                push    ebp
                mov     ebp,esp

                mov     eax,OFFSET WAV_DummyFunc
                mov     ds:dwUserFunc,eax

                $biosCall       WAV_OFFSET,60h  ; WAVE-BIOS 初期化

                movzx   eax,al

                leave
                ret
WAV_init        endp

include ..\inc\codeend.mac
```

wav61.asm

```
include ..\inc\codebgn.mac
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_end( ) ;
;*  Function    : ライブラリの終了
;*
;*                 [入力]
;*                   なし
;*
;*                 [戻値]
;*                   =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_end
WAV_end         proc    near
                push    ebp
                mov     ebp,esp

                $biosCall       WAV_OFFSET,61h  ; WAVE-BIOS 終了
```

```
                movzx   eax,al

                leave
                ret
WAV_end         endp
include ..\inc\codeend.mac
```

wav63.asm

```
include ..\inc\codebgn.mac
;*****************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*----------------------------------------------------------------
;*
;*  Entry  Name : int WAV_reset( ) ;
;*  Function    : 録音・再生状態の初期化
;*
;*                [入力]
;*                  なし
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;*****************************************************************
                align   4
                public  WAV_reset
WAV_reset       proc    near
                push    ebp
                mov     ebp,esp

                $biosCall       WAV_OFFSET,63h  ; リセット

                movzx   eax,al

                leave
                ret
WAV_reset       endp

include ..\inc\codeend.mac
```

wav64.asm

```
include ..\inc\codebgn.mac
;*****************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*----------------------------------------------------------------
;*
;*  Entry  Name : int WAV_getMute( ) ;
;*  Function    : 再生音量のミュート取得
;*
;*                [入力]
;*                  int  *mode : ミュートモード(=0:解除,=1:ミュート中)
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;*****************************************************************
                align   4
                public  WAV_getMute
WAV_getMute     proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edi

                mov     cl,2                    ; ミュート状態取得

                $biosCall       WAV_OFFSET,64h  ; 再生音量のミュート

                movzx   ebx,bl
                mov     edi,dword ptr PRM1
                mov     dword ptr [edi],ebx

                movzx   eax,al

                pop     edi
                pop     ecx
                pop     ebx
                leave
                ret
WAV_getMute     endp

include ..\inc\codeend.mac
```

wav64a.asm

```
include ..\inc\codebgn.mac
;*****************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*----------------------------------------------------------------
;*
;*  Entry  Name : int WAV_setMute( ) ;
;*  Function    : 再生音量のミュート設定
;*
;*                [入力]
;*                  int  mode : ミュートモード(=0:解除,=1:設定)
;*
;*                [戻値]
;*                  =  0 : 正常終了
```

```
;*
;**********************************************************************
                align   4
                public  WAV_setMute
WAV_setMute     proc    near
                push    ebp
                mov     ebp,esp
                push    ecx

                mov     ecx,dword ptr PRM1

                $biosCall       WAV_OFFSET,64h  ; 再生音量のミュート

                movzx   eax,al

                pop     ecx
                leave
                ret
WAV_setMute     endp

include ..\inc\codeend.mac
```

wav65.asm

```
include ..\inc\codebgn.mac
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_setVolume( l_vol, r_vol ) ;
;*  Function    : 再生音量設定
;*
;*                [入力]
;*                  int  l_vol : 左音量 (0～127)
;*                  int  r_vol : 右音量 (0～127)
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;**********************************************************************
                align   4
                public  WAV_setVolume
WAV_setVolume   proc    near
                push    ebp
                mov     ebp,esp
                push    ebx

                mov     bl,byte ptr PRM1        ; 左音量
                mov     bh,byte ptr PRM2        ; 右音量

                $biosCall       WAV_OFFSET,65h  ; 再生音量設定

                movzx   eax,al

                pop     ebx

                leave
                ret
WAV_setVolume   endp

include ..\inc\codeend.mac
```

wav66.asm

```
include ..\inc\codebgn.mac
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_getVolume( l_vol, r_vol ) ;
;*  Function    : 再生音量取得
;*
;*                [入力]
;*                  int  *l_vol : 左音量 (0～127)
;*                  int  *r_vol : 右音量 (0～127)
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;**********************************************************************
                align   4
                public  WAV_getVolume
WAV_getVolume   proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edi

                $biosCall       WAV_OFFSET,66h  ; 再生音量取得

                mov     edi,dword ptr PRM1      ; 左音量
                movzx   ecx,bl
                mov     dword ptr [edi],ecx

                mov     edi,dword ptr PRM2      ; 右音量
                mov     cl,bh
                mov     dword ptr [edi],ecx

                movzx   eax,al

                pop     edi
```

```
                pop     ecx
                pop     ebx

                leave
                ret
WAV_getVolume   endp

include ..\inc\codeend.mac
```

wav67.asm

```
include ..\inc\codebgn.mac
;***********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*----------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_getCapability( capa, freq ) ;
;*  Function    : 録音／再生性能取得
;*
;*                [入力]
;*                  int  *capa : 性能情報
;*                  int  freq  : サンプリング周波数（単位:Hz）
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;***********************************************************************
                align   4
                public  WAV_getCapability
WAV_getCapability       proc    near
                push    ebp
                mov     ebp,esp
                push    edx
                push    edi

                mov     edx,dword ptr PRM2
                $biosCall       WAV_OFFSET,67h  ; 録音／再生性能取得

                mov     edi,dword ptr PRM1
                mov     dword ptr [edi],edx

                movzx   eax,al

                pop     edi
                pop     edx

                leave
                ret
WAV_getCapability       endp

include ..\inc\codeend.mac
```

wav68.asm

```
include ..\inc\codebgn.mac
;***********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*----------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_getStatus( status ) ;
;*  Function    : 録音／再生状態取得
;*
;*                [入力]
;*                  int  *staus : 録音／再生状態
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;***********************************************************************
                align   4
                public  WAV_getStatus
WAV_getStatus   proc    near
                push    ebp
                mov     ebp,esp
                push    edx
                push    edi

                $biosCall       WAV_OFFSET,68h  ; 録音／再生状態取得

                mov     edi,dword ptr PRM1
                mov     dword ptr [edi],edx

                movzx   eax,al

                pop     edi
                pop     edx

                leave
                ret
WAV_getStatus   endp

include ..\inc\codeend.mac
```

wav69.asm

```
include ..\inc\codebgn.mac
;***********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*----------------------------------------------------------------------
;*
```

```
;*  Entry  Name : int WAV_setWaveInfo( buf,freq, bitno, kind, pcmsz,
;*                                     pcmst ) ;
;*  Function    : WAVE データ情報設定
;*
;*               [入力]
;*                 char *buf   : WAVE ヘッダを格納するバッファアドレス
;*                 int  freq   : サンプリング周波数(単位:Hz)
;*                 int  bitno  : サンプリングビット数(8 or 16)
;*                 int  kind   : データ種別(1:モノラル,2:ステレオ)
;*                 int  pcmsz  : PCM データ長
;*                 int  *pcmst : PCM データ格納開始相対位置
;*
;*               [戻値]
;*                 =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_setWaveInfo
WAV_setWaveInfo proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    edi
                push    es

                push    ds
                pop     es
                mov     edi,dword ptr PRM1      ; バッファアドレス

                mov     edx,dword ptr PRM2      ; サンプリング周波数
                mov     ch,byte ptr PRM3        ; サンプリングビット数
                mov     cl,byte ptr PRM4        ; データ種別
                mov     ebx,dword ptr PRM5      ; PCM データ長

                $biosCall       WAV_OFFSET,69h  ; WAVE データ情報設定

                mov     edi,dword ptr PRM6      ; PCM データ格納開始相対位置
                mov     dword ptr ds:[edi],edx

                movzx   eax,al

                pop     es
                pop     edi
                pop     edx
                pop     ecx
                pop     ebx

                leave
                ret
WAV_setWaveInfo endp
```

```
include ..\inc\codeend.mac
```

wav6a.asm

```
include ..\inc\codebgn.mac
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_getWaveInfo( buf, bufsz, freq, bitno, kind,
;*                                     pcmsz, pcmst ) ;
;*  Function    : WAVE ファイルの情報取得
;*
;*               [入力]
;*                 char *buf   : WAVE ヘッダが格納されているアドレス
;*                 int  bufsz  : buf サイズ
;*                 int  *freq  : サンプリング周波数(単位:Hz)
;*                 int  *bitno : サンプリングビット数(8 or 16)
;*                 int  *kind  : データ種別(1:モノラル,2:ステレオ)
;*                 int  *pcmsz : PCM データ長
;*                 int  *pcmst : PCM データ格納開始相対位置
;*
;*               [戻値]
;*                 =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_getWaveInfo
WAV_getWaveInfo proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    edi
                push    esi
                push    es

                push    ds
                pop     es
                mov     edi,dword ptr PRM1
                mov     ecx,dword ptr PRM2

                $biosCall       WAV_OFFSET,6Ah  ; WAVE データ情報取得

                mov     edi,dword ptr PRM3      ; サンプリング周波数
                mov     dword ptr [edi],edx

                mov     edi,dword ptr PRM4      ; サンプリングビット数
                movzx   edx,ch
                mov     dword ptr [edi],edx
```

```
                mov     edi,dword ptr PRM5      ; データ種別
                mov     dl,cl
                mov     dword ptr [edi],edx

                mov     edi,dword ptr PRM6      ; PCM データ長
                mov     dword ptr [edi],ebx

                mov     edi,dword ptr PRM7      ; PCM データ格納開始相対位置
                mov     dword ptr [edi],esi

                movzx   eax,al

                pop     es
                pop     esi
                pop     edi
                pop     edx
                pop     ecx
                pop     ebx

                leave
                ret
WAV_getWaveInfo endp

include ..\inc\codeend.mac
```

wav6b.asm

```
include ..\inc\codebgn.mac
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_makeTable( ringbuf, cntltbl ) ;
;*  Function    : リングバッファ管理テーブル作成
;*
;*                [入力]
;*                  char *ringbuf : リングバッファ用領域アドレス
;*                  char *cntltbl : リングバッファ管理テーブルアドレス
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_makeTable
WAV_makeTable   proc    near
                push    ebp
                mov     ebp,esp
                push    edi
                push    esi
                push    es

                push    ds
                pop     es
                mov     esi,dword ptr PRM1
                mov     edi,dword ptr PRM2

                $biosCall       WAV_OFFSET,6Bh  ;

                movzx   eax,al

                pop     es
                pop     esi
                pop     edi

                leave
                ret
WAV_makeTable   endp

include ..\inc\codeend.mac
```

wav6c.asm

```
include ..\inc\codebgn.mac
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_pushTable( ptr ) ;
;*  Function    : 現在のリングバッファ管理テーブル及びリングバッファの
;*                アドレスを退避
;*
;*                [入力]
;*                  char *ptr : 退避領域アドレス
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_pushTable
WAV_pushTable   proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    edi
                push    esi
                push    ds
                push    es
                push    fs
```

```
                push    ds
                pop     fs                          ; ds を fs に設定

                mov     bl,0
                $biosCall       WAV_OFFSET,6Ch  ; アドレス取得

                mov     ebx,dword ptr PRM1
                mov     dword ptr fs:[ebx+dwRingOff],esi
                mov      word ptr fs:[ebx+wRingSel],ds
                mov     dword ptr fs:[ebx+dwRingCtrlOff],edi
                mov      word ptr fs:[ebx+wRingCtrlSel],es

                movzx   eax,al

                pop     fs
                pop     es
                pop     ds
                pop     esi
                pop     edi
                pop     ebx

                leave
                ret
WAV_pushTable   endp

include ..\inc\codeend.mac
```

wav6ca.asm

```
include ..\inc\codebgn.mac
;*******************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_popTable( ptr ) ;
;*  Function    : 退避したリングバッファ管理テーブル及びリングバッファ
;*                のアドレスを復帰
;*
;*                [入力]
;*                  char *ptr : 退避領域アドレス
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;*******************************************************************
                align   4
                public  WAV_popTable
WAV_popTable    proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    edi
                push    esi
                push    ds
                push    es
                push    fs

                push    ds
                pop     fs                          ; ds を fs に設定

                mov     ebx,dword ptr PRM1
                lds     esi,pword ptr fs:[ebx+dwRingOff]
                les     edi,pword ptr fs:[ebx+dwRingCtrlOff]

                mov     bl,1
                $biosCall       WAV_OFFSET,6Ch  ; アドレス退避

                movzx   eax,al

                pop     fs
                pop     es
                pop     ds
                pop     esi
                pop     edi
                pop     ebx

                leave
                ret
WAV_popTable    endp

include ..\inc\codeend.mac
```

wav70.asm

```
include ..\inc\codebgn.mac
                extrn   WAV_DummyFunc:near
                extrn   WAV_CallBack:far

                extrn   dwUserFunc:dword
                extrn   dwParamTable:dword
                extrn   dwStack_Bottom:dword
;*******************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_recPrepare( freq, bitno, kind, recbuf,
;*                                    syncFunc ) ;
;*  Function    : 録音前準備
;*
;*                [入力]
;*                  int  freq     : サンプリング周波数（単位:Hz）
;*                  int  bitno    : サンプリングビット数（8 or 16）
```

```
;*              int  kind     : データ種別(1:モノラル,2:ステレオ)
;*              char *recbuf  : 録音データを格納するアドレス
;*              void syncFunc : 録音処理と同期して行う処理関数
;*
;*          [戻値]
;*            =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_recPrepare
WAV_recPrepare  proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    edi
                push    es

                mov     edx,dword ptr PRM1
                mov     ch,byte ptr PRM2
                mov     cl,byte ptr PRM3
                mov     ebx,dword ptr PRM4
                mov     edi,dword ptr PRM5
                or      edi,edi                 ; 割り込み関数は NULL?
                jnz     short #Set_UserFunc     ; No, Jump #Set_UserFunc

                mov     edi,OFFSET WAV_DummyFunc

#Set_UserFunc:
                mov     ds:dwUserFunc,edi

                push    ds
                pop     es
                mov     edi,OFFSET dwParamTable

                mov     [edi].dwDataOffset,ebx
                mov     [edi].wDataSelector,ds
                mov     ebx,OFFSET WAV_CallBack
                mov     [edi].dwFuncOffset,ebx
                mov     [edi].wFuncSelector,cs
                mov     ebx,OFFSET dwStack_Bottom
                mov     [edi].dwStackOffset,ebx
                mov     [edi].wStackSelector,ds
                mov     [edi].wFuncDS,ds
                mov     [edi].wFuncES,es
                mov     [edi].wFuncFS,fs
                mov     [edi].wFuncGS,gs

                $biosCall       WAV_OFFSET,70h  ; 録音前準備

                movzx   eax,al

                pop     es
                pop     edi
                pop     edx
                pop     ecx
                pop     ebx

                leave
                ret
WAV_recPrepare  endp

include ..\inc\codeend.mac
```

wav71.asm

```
include ..\inc\codebgn.mac
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_rec( mode, trig, bufsz ) ;
;*  Function    : 録音開始
;*
;*          [入力]
;*            int           mode  : 録音モード
;*            int           trig  : トリガレベル(0~127)
;*            unsigned int  bufsz : サンプリングデータ長
;*
;*          [戻値]
;*            =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_rec
WAV_rec         proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx

                mov     bh,byte ptr PRM1
                mov     bl,byte ptr PRM2
                mov     ecx,dword ptr PRM3

                $biosCall       WAV_OFFSET,71h  ; 録音開始

                movzx   eax,al

                pop     ecx
                pop     ebx
```

```
                leave
                ret
WAV_rec         endp

include ..\inc\codeend.mac
```

wav72.asm

```
include ..\inc\codebgn.mac
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*---------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_recStop( ) ;
;*  Function    : 録音強制終了
;*
;*                [入力]
;*                  なし
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;**********************************************************************
                align   4
                public  WAV_recStop
WAV_recStop     proc    near
                push    ebp
                mov     ebp,esp

                $biosCall       WAV_OFFSET,72h  ; 録音強制終了

                movzx   eax,al

                leave
                ret
WAV_recStop     endp

include ..\inc\codeend.mac
```

wav73.asm

```
include ..\inc\codebgn.mac
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*---------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_recGetAddress( buf ) ;
;*  Function    : 録音データの格納アドレスの取得
;*
;*                [入力]
;*                  char *buf : 録音データアドレスの取得バッファ
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;**********************************************************************
                align   4
                public  WAV_recGetAddress
WAV_recGetAddress       proc    near
                push    ebp
                mov     ebp,esp
                push    esi
                push    edi
                push    es

                $biosCall       WAV_OFFSET,73h  ; アドレス取得

                mov     esi,dword ptr PRM1
                mov     dword ptr ds:[esi+0],edi
                mov      word ptr ds:[esi+4],es

                movzx   eax,al

                pop     es
                pop     edi
                pop     esi

                leave
                ret
WAV_recGetAddress       endp

include ..\inc\codeend.mac
```

wav78.asm

```
include ..\inc\codebgn.mac
                extrn   WAV_DummyFunc:near
                extrn   WAV_CallBack:far

                extrn   dwUserFunc:dword
                extrn   dwParamTable:dword
                extrn   dwStack_Bottom:dword
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*---------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_playPrepare( freq, bitno, kind, playbuf,
;*                                     syncFunc ) ;
;*  Function    : 再生前準備
```

```
;*
;*              [入力]
;*                int  freq     : サンプリング周波数(単位:Hz)
;*                int  bitno    : サンプリングビット数(8 or 16)
;*                int  kind     : データ種別(1:モノラル,2:ステレオ)
;*                char *playbuf : 再生データの先頭アドレス
;*                void syncFunc : 再生処理と同期して行う処理関数
;*
;*              [戻値]
;*                = 0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_playPrepare
WAV_playPrepare proc    near
                push    ebp
                mov     ebp,esp
                push    ebx
                push    ecx
                push    edx
                push    edi
                push    es

                mov     edx,dword ptr PRM1
                mov     ch,byte ptr PRM2
                mov     cl,byte ptr PRM3
                mov     ebx,dword ptr PRM4
                mov     edi,dword ptr PRM5
                or      edi,edi                 ; 割り込み関数は NULL?
                jnz     short #Set_UserFunc     ; No, Jump #Set_UserFunc

                mov     edi,OFFSET WAV_DummyFunc

#Set_UserFunc:
                mov     ds:dwUserFunc,edi

                push    ds
                pop     es
                mov     edi,OFFSET dwParamTable

                mov     [edi].dwDataOffset,ebx
                mov     [edi].wDataSelector,ds
                mov     ebx,OFFSET WAV_CallBack
                mov     [edi].dwFuncOffset,ebx
                mov     [edi].wFuncSelector,cs
                mov     ebx,OFFSET dwStack_Bottom
                mov     [edi].dwStackOffset,ebx
                mov     [edi].wStackSelector,ds
                mov     [edi].wFuncDS,ds
                mov     [edi].wFuncES,es
                mov     [edi].wFuncFS,fs
                mov     [edi].wFuncGS,gs
                $biosCall       WAV_OFFSET,78h  ; 再生前準備

                movzx   eax,al

                pop     es
                pop     edi
                pop     edx
                pop     ecx
                pop     ebx

                leave
                ret
WAV_playPrepare endp

include ..\inc\codeend.mac
```

wav79.asm

```
include ..\inc\codebgn.mac
;*********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*-------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_play( bufsz ) ;
;*  Function    : 再生開始
;*
;*              [入力]
;*                unsigned int  bufsz : サンプリングデータ長
;*
;*              [戻値]
;*                =  0 : 正常終了
;*
;*********************************************************************
                align   4
                public  WAV_play
WAV_play        proc    near
                push    ebp
                mov     ebp,esp
                push    ecx

                mov     ecx,dword ptr PRM1

                $biosCall       WAV_OFFSET,79h  ; 再生開始

                movzx   eax,al

                pop     ecx

                leave
                ret
```

```
WAV_play        endp

include ..\inc\codeend.mac
```

wav7a.asm

```
include ..\inc\codebgn.mac
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_playStop( ) ;
;*  Function    : 再生強制終了
;*
;*                [入力]
;*                  なし
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;**********************************************************************
                align   4
                public  WAV_playStop
WAV_playStop    proc    near
                push    ebp
                mov     ebp,esp

                $biosCall       WAV_OFFSET,7Ah  ; 再生強制終了

                movzx   eax,al

                leave
                ret
WAV_playStop    endp

include ..\inc\codeend.mac
```

wav7b.asm

```
include ..\inc\codebgn.mac
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : int WAV_playGetAddress( buf ) ;
;*  Function    : 再生データアドレスの取得
;*
;*                [入力]
;*                  char *buf : 再生データアドレスの取得バッファ
;*
;*                [戻値]
;*                  =  0 : 正常終了
;*
;**********************************************************************
                align   4
                public  WAV_playGetAddress
WAV_playGetAddress      proc    near
                push    ebp
                mov     ebp,esp
                push    esi
                push    edi
                push    es

                $biosCall       WAV_OFFSET,7Bh  ; アドレス取得

                mov     esi,dword ptr PRM1
                mov     dword ptr ds:[esi+0],edi
                mov      word ptr ds:[esi+4],es

                movzx   eax,al

                pop     es
                pop     edi
                pop     esi

                leave
                ret
WAV_playGetAddress      endp

include ..\inc\codeend.mac
```

wavdummy.asm

```
include ..\inc\codebgn.mac
                extrn   dwUserFunc:dword
;**********************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*--------------------------------------------------------------------
;*
;*  Entry  Name : void WAV_DummyFunc( ) ;
;*  Function    : コールバックダミー関数
;*
;*                [入力]
;*                  無し
;*
;*                [戻値]
;*                  無し
;*
;**********************************************************************
                align   4
```

```
                public  WAV_DummyFunc
WAV_DummyFunc   proc    near
                ret
WAV_DummyFunc   endp

;*******************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*------------------------------------------------------------------
;*
;*  Entry  Name : void WAV_DummyFunc( ) ;
;*  Function    : コールバック関数を起動する
;*
;*                 [入力]
;*                   無し
;*
;*                 [戻値]
;*                   無し
;*
;*******************************************************************
                align   4
                public  WAV_CallBack
WAV_CallBack    proc    far
                call    cs:dwUserFunc
                ret
WAV_CallBack    endp

include ..\inc\codeend.mac
```

wavdata.asm

```
page    ,132
;*******************************************************************
;*  Wave BIOS C Linrary for Towns OS V2.1 L30 or later
;*------------------------------------------------------------------
;*
;*  Program Name : WAV.LIB
;*  File    Name : WAVDATA.ASM
;*  Function     : Wave BIOS Cライブラリ
;*
;*******************************************************************
include ..\inc\wavlib.h
;*******************************************************************
;       ライブラリ内部データ領域の定義
;*******************************************************************
                .386p
                assume  cs:WAV_code,ds:WAV_data
CGROUP          group   WAV_code
WAV_code        segment dword public 'CODE' use32
WAV_code        ends
DGROUP          group   WAV_data
WAV_data        segment dword public 'DATA' use32

                extrn   WAV_DummyFunc:near
                public  dwParamTable,dwUserFunc,dwStack_Bottom

                align   4
dwParamTable    PREPAREPRM     <>              ; Parameter Table

                align   4
dwUserFunc      DD      WAV_DummyFunc          ; CallBack Function

                align   4
dwUserStack     DB      1024    dup(0FEh)      ; Stack for CallBack Function
dwStack_Bottom  LABEL   BYTE

WAV_data        ends

                end
```

C.9 音源割り込み管理BIOSサンプル

int_05.asm

```
include codebgn.mac

                public  MOS_getTime
MOS_getTime     proc    near
                $biosCall       INT_OFFSET,05h
                ret
MOS_getTime     endp

include codeend.mac
```

int_08.asm

```
include codebgn.mac
;---------------------------------------------------------------------------
;  マウスイベントルーチンの取得2
; MOS_getEvent2(char *buf)
; この関数では、ＴＢＩＯＳがイベントを呼ぶ際に
; 設定するセグメント値と仲介ルーチンのアドレス
; を返します。
```

```
;--------------------------------------------------------------------------
MOS_eventnum    equ     003h
;--------------------------------------------------------------------------
                public  MOS_getEvent2
MOS_getEvent2   proc    near

                push    esi
                mov     esi,ss:[esp+8]
                mov     al,MOS_eventnum
                pushfd
                cli
                $biosCall       INT_OFFSET,08h
                popfd
                pop     esi
                ret

MOS_getEvent2   endp

include codeend.mac
```

int_09.asm

```
include codebgn.mac

                public  MOS_getIntStatus
MOS_getIntStatus        proc    near
                $biosCall       INT_OFFSET,09h
                ret
MOS_getIntStatus        endp

include codeend.mac
```

INT.asm

```
        page    ,132
;--------------------------------------------------------------------------
;
;      ＴＢＩＯＳ割り込み管理ＢＩＯＳ　サンプルプログラム
;
;
;
;      機能：割り込み管理ＢＩＯＳを使用して、音声出力とマウスの制御を行います
;
;--------------------------------------------------------------------------
;
;       assemble usage: 386asm intsmpl -twocase
;
;       link usage:     386link intsmpl -twocase -stack 8192 -pack
;
;--------------------------------------------------------------------------
                .386p
                assume  cs:SAMPLE_code,ds:SAMPLE_data
CGROUP          group   SAMPLE_code
SAMPLE_code    segment dword public 'CODE' use32
;--------------------------------------------------------------------------
EgbWorkSize     equ     1536                    ; EGB work size
MosWorkSize     equ     4096                    ; MOS work size
SndWorkSize     equ     16384                   ; SND work size
TBIOS_CSEG      equ     0110h                   ; ＴＢＩＯＳセレクタ番号
INT_TYPE_SOUND  equ     04dh                    ; interrupt level
TIMER_B_SND     equ     0ddh                    ; SND mouse counter (10.08ms)
TIMER_B_MOS     equ     0b0h                    ; MOS mouse counter (23.04ms)
PCMPLAY_CH      equ     71                      ; 音声再生チャンネル番号
;--------------------------------------------------------------------------
;       メイン
;--------------------------------------------------------------------------
                align   4
main            proc    near
                pushfd
                pushad
                push    ds
                push    es
                push    fs
                push    gs
                cld                             ; direction flag clear
                push    ds
                pop     gs                      ; gs = ds
                push    dword ptr TBIOS_CSEG
                pop     fs                      ; fs = TBIOS code seg
#EGB_setup:
                sub     ah,ah
                mov     ecx,EgbWorkSize
                call    EGB_BIOS                ; ＥＧＢ初期化
                mov     ax,0600h
                mov     edx,1
                call    EGB_BIOS                ; 表示ページの指定
#SND_setup:
                call    SNDINT_start            ; ＳＮＤ初期化
                mov     ah,46h
                mov     bl,00000001b
                call    SND_BIOS                ; 電子ボリュームミュート
                mov     ah,21h
                mov     bl,1
                call    SND_BIOS                ; 音声モードチャンネルの設定
                call    GET_NUMBER
                call    PLAY_ROM_VOICE          ;現在の時間に合う音声
                call    PLAY_CHECK
#MOS_setup:
                call    MOSINT_start            ;ＭＯＳ初期化
                call    MOSEVENT_start          ;マウスイベント登録
                mov     ah,002h
                mov     al,001h
```

```
                call    MOS_BIOS                ; マウス表示
                mov     eax,8
                call    PLAY_ROM_VOICE          ;「入力してください」
                call    PLAY_CHECK
#mouse_sense:
                mov     ah,003h
                call    MOS_BIO  S              ; 位置とボタンの読み取り
                and     ch,00000011b
                cmp     ch,00000011b            ; 左右ボタンが押された？
                jne     short #mouse_sense
#MOS_end:
                call    MOSEVENT_end            ; マウスイベント終了
                call    MOSINT_end              ; ＭＯＳ終了
#Good_bye_message:
                mov     eax,5
                call    PLAY_ROM_VOICE          ;「さようなら」
                call    PLAY_CHECK
#SND_end:
                call    SNDINT_end              ; ＳＮＤ終了
#exit:
                pop     gs
                pop     fs
                pop     es
                pop     ds
                popad
                popfd
                mov     ax,4c00h                ; Program terminate.
                int     21h
main            endp
;------------------------------------------------------------------------------
;       現在の時間用の音声の番号を得る
;       in:     無し
;       out:    eax = 番号
;------------------------------------------------------------------------------
                align   4
GET_NUMBER      proc    near
                mov     ah,2ch
                int     21h                     ; システム時間の入手
                movzx   ebx,ch
                movzx   eax,byte ptr cs:[ebx+#rom_number]
                ret
#rom_number     label   byte
                db      003h,003h,003h,003h,003h,001h   ; 00:??～05:??
                db      001h,001h,001h,001h,001h,002h   ; 06:??～11:??
                db      002h,002h,002h,002h,002h,002h   ; 12:??～17:??
                db      003h,003h,003h,003h,003h,003h   ; 18:??～23:??
GET_NUMBER      endp
;------------------------------------------------------------------------------
;       システムＲＯＭの音声を再生する
;       in:     eax = 番号
;       out:    無し
;       +----+--------------------------+
;           | 01 |  おはよう
;           | 02 |  こんにちは
;           | 03 |  こんばんは
;           | 04 |  バイバイ
;           | 05 |  さようなら
;           | 06 |  おやすみ
;           | 07 |  選んでください
;           | 08 |  入力してください
;           | 09 |  ごめんなさい
;           | 10 |  確認してください
;           +----+--------------------------+
;------------------------------------------------------------------------------
VOICE_SEG       equ     148h
VOICE_OFFSET    equ     40000h
;------------------------------------------------------------------------------
                align   4
PLAY_ROM_VOICE  proc    near
                push    ds
                push    dword ptr VOICE_SEG
                pop     ds
                mov     esi,VOICE_OFFSET
                cmp     ds:[esi],eax            ; その番号の音声は存在する？
                jb      short #exit
                test    eax,eax                 ; 音声番号 = 0 ？
                jz      short #exit
                mov     eax,[esi+eax*4]
                add     esi,eax
                mov     ah,25h
                mov     bl,PCMPLAY_CH
                mov     dh,62
                mov     dl,127
                call    SND_BIOS                ; 音声モードＰＣＭ再生
#exit:
                pop     ds
                ret
PLAY_ROM_VOICE  endp
;------------------------------------------------------------------------------
;       音声再生終了チェックと音声停止
;       in:     無し
;       out:    無し
;------------------------------------------------------------------------------
                align   4
PLAY_CHECK      proc    near
#loop:
                mov     bl,PCMPLAY_CH
                mov     ah,28h
                call    SND_BIOS                ; 音声モードＰＣＭ再生状態
                test    dl,dl
                jnz     short #loop             ; 音声出力が終了するまでループ
                mov     ah,27h
                mov     bl,PCMPLAY_CH
                call    SND_BIOS                ; 音声モードＰＣＭ再生中断
```

```
                ret
PLAY_CHECK      endp
;-----------------------------------------------------------------------------
;       マウスイベントの登録
;       in:     無し
;       out:    無し
;-----------------------------------------------------------------------------
                align   4
MOSEVENT_start  proc    near
                push    ds
                push    esi
                pushfd
                cli
                push    gs
                push    fs
                push    es
                push    ds
                push    cs
                push    offset EVENT_main
                mov     esi,esp
                push    ss
                pop     ds
                mov     ah,006h
                mov     al,003h
                call    INT_BIOS                ; 割り込み／割り出し処理の登録
                add     esp,24
                popfd
                pop     esi
                pop     ds
                ret
MOSEVENT_start_endp
;-----------------------------------------------------------------------------
;       イベントルーチン
;-----------------------------------------------------------------------------
                align   4
EVENT_main      proc    far
                push    fs
                push    gs
                push    esi
                push    edi
                push    ds
                pop     gs                      ; gs = ds
                push    dword ptr TBIOS_CSEG
                pop     fs                      ; fs = TBIOS code seg
                mov     ah,005h .
                call    INT_BIOS                ; 割り込み動作回数の読み取り
                test    al,00001111b
                jnz     short #exit
                mov     esi,offset PAL_WORK
                mov     ds:[esi+08],al
                mov     ds:[esi+09],al
                mov     ds:[esi+10],al
                mov     ah,004h
                sub     al,al
                call    EGB_BIOS                ; パレットの設定
#exit:
                pop     edi
                pop     esi
                pop     gs
                pop     fs
                ret
EVENT_main      endp
;-----------------------------------------------------------------------------
;
;-----------------------------------------------------------------------------
                align   4
MOSEVENT_end    proc    near
                mov     ah,007h
                mov     al,003h
                call    INT_BIOS                ; 割り込み／割り出し処理の解除
                ret
MOSEVENT_end    endp
;-----------------------------------------------------------------------------
;       E G B   C a l l
;-----------------------------------------------------------------------------
EGB_OFFSET      equ     0020h
;-----------------------------------------------------------------------------
                align   4
EGB_BIOS        proc    near
                mov     edi,offset EGB_WORK
                call    pword ptr fs:[EGB_OFFSET]
                ret
EGB_BIOS        endp
;-----------------------------------------------------------------------------
;       S N D   C a l l
;-----------------------------------------------------------------------------
SND_OFFSET      equ     0080h
;-----------------------------------------------------------------------------
                align   4
SND_BIOS        proc    near
                mov     edi,offset SND_WORK
                call    pword ptr fs:[SND_OFFSET]
                ret
SND_BIOS        endp
;-----------------------------------------------------------------------------
;       M O S   C a l l
;-----------------------------------------------------------------------------
MOS_OFFSET      equ     0040h
;-----------------------------------------------------------------------------
                align   4
MOS_BIOS        proc    near
                mov     edi,offset MOS_WORK
                call    pword ptr fs:[MOS_OFFSET]
                ret
MOS_BIOS        endp
;-----------------------------------------------------------------------------
```

```
;       ＴＢＩＯＳ割り込み管理ＢＩＯＳ　Ｃａｌｌ
;-----------------------------------------------------------------------------
INT_OFFSET      equ     01a0h
;-----------------------------------------------------------------------------
                align   4
INT_BIOS        proc    near
                call    pword ptr fs:[INT_OFFSET]
                ret
INT_BIOS        endp
;-----------------------------------------------------------------------------
;       システム情報ＢＩＯＳ　Ｃａｌｌ
;-----------------------------------------------------------------------------
SYS_OFFSET      equ     01c0h
;-----------------------------------------------------------------------------
                align   4
SYS_BIOS        proc    near
                call    pword ptr fs:[SYS_OFFSET]
                ret
SYS_BIOS        endp
;-----------------------------------------------------------------------------
;       タイマＢへの設定と始動
;       in:     dl = タイマＢ設定値
;       out:    無し
;-----------------------------------------------------------------------------
                align   4
TIMERB_sub      proc    near
                mov     ah,031h
                sub     bh,bh
                mov     dh,026h
                call    SND_BIOS
                mov     ah,031h
                sub     bh,bh
                mov     dx,0272ah
                call    SND_BIOS
                ret
TIMERB_sub      endp
;-----------------------------------------------------------------------------
;       ＳＮＤ割り込み処理動作開始
;-----------------------------------------------------------------------------
                align   4
SNDINT_start    proc    near
                sub     ah,ah
                call    SND_BIOS                ; ＳＮＤ初期化
                pushfd
                cli
                mov     ah,003h
                mov     edx,offset STACK_BOTTOM
                call    INT_BIOS                ; ＳＮＤ割り込み処理の登録
                test    eax,eax
                jz      short #timer_set
                call    EXTENDER_start          ; DOS-Extender への設定
#timer_set:
                mov     dl,TIMER_B_SND
                call    TIMERB_sub              ; timer B start
                popfd
                ret
SNDINT_start    endp
;-----------------------------------------------------------------------------
;       ＭＯＳ割り込み処理動作開始
;-----------------------------------------------------------------------------
                align   4
MOSINT_start    proc    near
                sub     ah,ah
                mov     ecx,MosWorkSize
                call    MOS_BIOS                ; ＭＯＳ初期化
                pushfd
                cli
                mov     ah,001h
                mov     edx,offset STACK_BOTTOM
                call    INT_BIOS                ; ＭＯＳ割り込み処理の登録
                mov     dl,TIMER_B_SND
                test    eax,eax
                jz      short #timer_set
                call    EXTENDER_start          ; DOS-Extender への設定
                mov     dl,TIMER_B_MOS
#timer_set:
                call    TIMERB_sub              ; timer B start
                popfd
                ret
MOSINT_start    endp
;-----------------------------------------------------------------------------
;       DOS-Extender への割り込み処理登録のための設定
;-----------------------------------------------------------------------------
INTERRUPT       equ     01a8h
;-----------------------------------------------------------------------------
                align   4
EXTENDER_start  proc    near
                mov     cl,INT_TYPE_SOUND
                mov     ax,02502h
                int     21h
                jc      short #exit
                mov     ah,30h
                mov     al,1
                mov     edx,ebx
                call    SYS_BIOS                ; save native offset
                mov     ah,30h
                mov     al,2
                sub     edx,edx
                mov     dx,es
                call    SYS_BIOS                ; save native segment
#get_real_bector:
                mov     cl,INT_TYPE_SOUND
                mov     ax,02503h
                int     21h
                jc      short #exit
```

```
                mov     ah,30h
                mov     al,3
                movzx   edx,bx
                call    SYS_BIOS                ; save real offset
                mov     ah,30h
                mov     al,4
                shr     ebx,16
                mov     edx,ebx
                call    SYS_BIOS                ; save real segment
                push    ds
                mov     cl,INT_TYPE_SOUND
                lds     edx,pword ptr fs:[INTERRUPT]
                mov     ax,02506h               ; 割り込みベクタの登録
                int     21h
                pop     ds
#exit:
                ret
EXTENDER_start  endp
;-----------------------------------------------------------------------------
;       ＳＮＤ割り込み処理動作終了
;-----------------------------------------------------------------------------
                align   4
SNDINT_end      proc    near
                sub     ah,ah
                call    SND_BIOS                ; ＳＮＤ初期化
                pushfd
                cli
                sub     bh,bh
                mov     dx,2c00h
                mov     ah,11h
                call    SND_BIOS                ; test register off
                mov     ah,004h
                call    INT_BIOS                ; ＳＮＤ割り込み処理の解除
                test    eax,eax
                jz      short #timer_set        ; ＭＯＳ割り込み処理動作中？
                call    EXTENDER_end
#exit:
                popfd
                ret
#timer_set:
                mov     dl,TIMER_B_MOS
                call    TIMERB_sub              ; timer B start
                jmp     short #exit
SNDINT_end      endp
;-----------------------------------------------------------------------------
;       ＭＯＳ割り込み処理動作終了
;-----------------------------------------------------------------------------
                align   4
MOSINT_end      proc    near
                mov     ah,1
                call    MOS_BIOS                ; ＭＯＳ終了
                pushfd
                cli
                mov     ah,002h
                call    INT_BIOS                ; ＭＯＳ割り込み処理の解除
                test    eax,eax
                jz      short #timer_set        ; ＳＮＤ割り込み処理動作中？
                call    EXTENDER_end
#exit:
                popfd
                ret
#timer_set:
                mov     dl,TIMER_B_SND
                call    TIMERB_sub              ; timer B start
                jmp     short #exit
MOSINT_end      endp
;-----------------------------------------------------------------------------
;       DOS-Extender への割り込み処理解除のための設定
;-----------------------------------------------------------------------------
                align   4
EXTENDER_end    proc    near
                push    ds
                mov     ah,31h
                mov     al,4
                call    SYS_BIOS                ; read real segment
                shl     edx,16
                push    edx
                mov     ah,31h
                mov     al,3
                call    SYS_BIOS                ; real real offset
                pop     ebx
                add     ebx,edx                 ; ebx = real offset | segment
                mov     ah,31h
                mov     al,2
                call    SYS_BIOS                ; read native segment
                mov     ds,dx
                mov     ah,31h
                mov     al,1
                call    SYS_BIOS                ; read native offset
                mov     cl,INT_TYPE_SOUND
                mov     ax,02507h               ; 割り込みベクタの設定
                int     21h
                pop     ds
                ret
EXTENDER_end    endp
;-----------------------------------------------------------------------------
SAMPLE_code     ends
DGROUP          group   SAMPLE_data
SAMPLE_data     segment dword public 'DATA' use32
;-----------------------------------------------------------------------------
                align   4
PAL_WORK        dd      1                       ; 設定パレット数
                dd      15                      ; 色識別番号
                db      4 dup(0)                ; 色データ
EGB_WORK        db      EgbWorkSize dup(0)      ; ＥＧＢワークエリア
MOS_WORK        db      MosWorkSize dup(0)      ; ＭＯＳワークエリア
```

```
SND_WORK        db      SndWorkSize dup(0)      ; ＳＮＤワークエリア
INT_STACK       db      1024 dup(?)             ; 割り込み処理用スタック
STACK_BOTTOM    label   byte
;-----------------------------------------------------------------------------
SAMPLE_data     ends
end
```

付録D

コード表

D.1 ASCII(7ビット)コード表

	0		1		2	3	4	5	6	7
0	↑@		↑P		(SP)	0	@	P	‘	p
1	↑A		↑Q	DUP	!	1	A	Q	a	q
2	↑B		↑R	INS	″	2	B	R	b	r
3	↑C		↑S		#	3	C	S	c	s
4	↑D		↑T		$	4	D	T	d	t
5	↑E	EL	↑U		%	5	E	U	e	u
6	↑F		↑V		&	6	F	V	f	v
7	↑G		↑W		’	7	G	W	g	w
8	↑H	⇦	↑X		(	8	H	X	h	x
9	↑I	TAB	↑Y		)	9	I	Y	i	y
A	↑J		↑Z		*	:	J	Z	j	z
B	↑K	HOME	↑〔	ESC	+	;	K	〔	k	{
C	↑L	CLS	↑¥	→	,	<	L	¥	l	|
D	↑M		↑〕	←	—	=	M	〕	m	}
E	↑N		↑^	↑	.	>	N	^	n	‾
F	↑O		↑—	↓	/	?	O	—	o	DL

コード00H～1FHの左側に示してある↑はコントロールキー(CTRL)との併用を示す.
例えば，↑AはCRTLを押しながらAのキーを押すことを示す.
コード80H～FFHは出力されません.

D.2 JIS(8ビット)コード表

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	↑@	↑P	(SP)	0	@	P	‘	p				ー	タ	ミ		
1	↑A	↑Q DUP	!	1	A	Q	a	q			。	ア	チ	ム		
2	↑B	↑R INS	″	2	B	R	b	r			「	イ	ツ	メ		
3	↑C	↑S	#	3	C	S	c	s			」	ウ	テ	モ		
4	↑D	↑T	$	4	D	T	d	t			、	エ	ト	ヤ		
5	↑E EL	↑U	%	5	E	U	e	u			・	オ	ナ	ユ		
6	↑F	↑V	&	6	F	V	f	v			ヲ	カ	ニ	ヨ		
7	↑G	↑W	’	7	G	W	g	w			ァ	キ	ヌ	ラ		
8	↑H ⇦	↑X	(	8	H	X	h	x			ィ	ク	ネ	リ		
9	↑I TAB	↑Y	)	9	I	Y	i	y			ゥ	ケ	ノ	ル		
A	↑J	↑Z	*	:	J	Z	j	z			ェ	コ	ハ	レ		
B	↑K HOME	↑〔 ESC	+	;	K	〔	k	{			ォ	サ	ヒ	ロ		
C	↑L CLS	↑¥ →	,	<	L	¥	l	|			ャ	シ	フ	ワ		
D	↑M	↑〕 ←	−	=	M	〕	m	}			ュ	ス	ヘ	ン		
E	↑N	↑^ ↑	.	>	N	^	n	￣			ョ	セ	ホ	゛		
F	↑O	↑— ↓	/	?	O	—	o	DL			ッ	ソ	マ	゜		

コード00H～1FHの左側に示してある↑はコントロールキー(CTRL)との併用を示す.
例えば，↑AはCTRLを押しながらAのキーを押すことを示す.
コード80H～9FHと0E0H～0FFHは，シフトJIS漢字コードの第1バイトのコードとして使用している.

D.3 特殊キーコード表

KEY	コード	KEY	コード	KEY	コード
PF1	8001	取消	8011	PF13	8021
PF2	8002	実行	8012	PF14	8022
PF3	8003	漢字辞書	8013	PF15	8023
PF4	8004	単語抹消	8014	PF16	8024
PF5	8005	単語登録	8015	PF17	8025
PF6	8006	前行	8016	PF18	8026
PF7	8007	次行	8017	PF19	8027
PF8	8008	半角/全角	8018	PF20	8028
PF9	8009				
PF10	800A	かな漢字	801C		
PF11	800B	PF12	801D		
		変換	801E		
		無変換	801F		

付録 E

80486CPUの概要

FM TOWNS II HR から，CPU には 80486(SX/DX) が搭載されています．80486 は，80386 命令実行の高速化と，キャッシュメモリ，80387(FPU) 相当機能の内蔵を図り，マルチプロセッサ支援機能を付加したものと評価できます．

このため，基本的に，80386 用に作ったソフトは 80486 でも走ります．言い換えると，386 搭載機用のソフトは，ほとんどが 486 搭載機でも使えます．しかし，一部のソフトには適合できないものもあります．

ここでは，486 の概要について述べるとともに，ソフトの不適合対策についてもヒントを提供したいと思います．

なお，ハードウェアの増設ピンについては，説明を省略します．

E.1 80486の強化ポイント

386 に対比して 486 で強化，改善された項目について，機能ごとに説明します．

E.1.1 キャッシュメモリ

内蔵キャッシュメモリを使うと，それが CPU と同じチップ内にあるということだけで高速化につながっています．なぜなら，キャッシュの内容が使える限り外部のバスと信号をやりとりしながら，データをアクセスする必要がないからです．

加えて，486 では，128 ビットバス (16 バイト) で内蔵キャッシュメモリから命令フェッチが行われるため，外部の 32 ビットバス (4 バイト) から直接フェッチするのに比べて格段に速くなります．

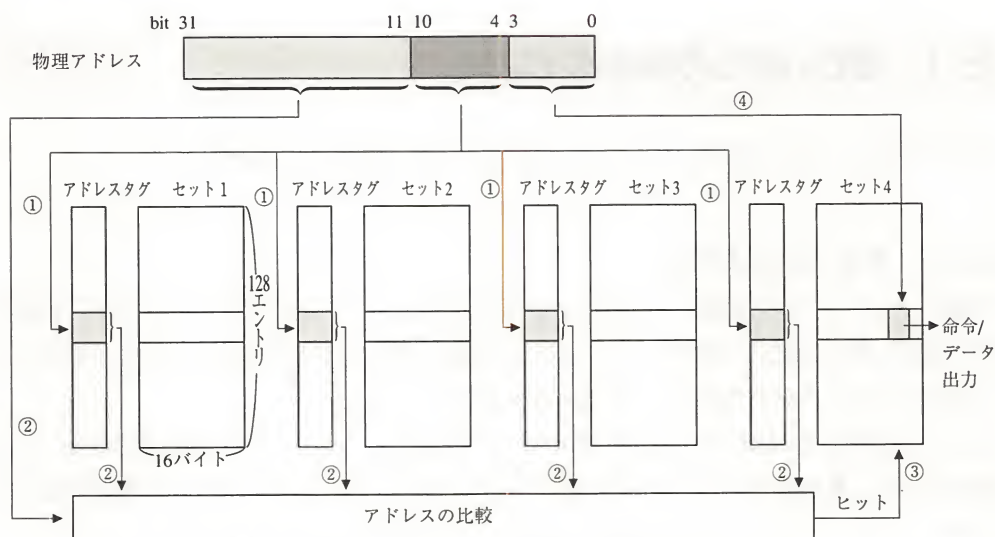
キャッシュが有効に働く条件は，CPU が処理したい命令やデータがその中にあるかどうかにかかっており，うまく存在したとき，これを「ヒット」したといいます．もしヒットしないと，CPU は外部のメモリの該当アドレスから一定サイズだけ読み出した内容をキャッシュに転送し

て，その内容で処理を続けます．このためには，キャッシュの中に空きが必要で，もしなければ(通常はプログラムの実行開始後すぐに空きがなくなる)，既存の命令やデータの一部を捨てなければなりません．捨てる対象は，486の場合，最も古くアクセスされたものとし，最近アクセスされたものは極力残そうとします．これは，最近のものの方が，今後再度アクセスされる可能性が高いからです．

さらに，486では，8KBのキャッシュを分割して，2KB単位の4面連動キャッシュ(4ウェイ・セット・アソシアティブ・キャッシュ)にしています(図E–1)．そして，その内容も，命令とデータの混合(ユニファイド・キャッシュ)とし，データアクセスの多いプログラムではデータが多く残り，その反対のプログラムでは命令が多く残るようにしています．こうすることによって，ヒット率が高くなります．

さて，キャッシュへの書き込みは，その内容が存在していたメモリへの書き込みを意味します．その場合，キャッシュとメモリの双方に書き込みする方法を「ライトスルー」と呼びます．メモリにだけ書いて，キャッシュに書き込みしない方法もあり，その場合はキャッシュ該当部分の内容は無効となります．

486では，キャッシュを操作する命令が追加されています．また，CPU内のレジスタで，キャッシュに関係するビットが追加されたものがあります．

▼図E-1　80486内蔵キャッシュの構造

E.1.2　FPU内蔵のメリット

386では外部にあったFPUが，486で内蔵化されることによって，キャッシュ同様，アクセ

スが速くなります.

また，メモリとFPU間のデータ転送時は，386のときはCPUが介在してデータを中継していましたが，486ではFPUとキャッシュメモリとの間を64ビットのバスで結び，倍精度データを直接転送することができるようになっています．このため，FPUを使う演算も速度が大幅に向上しています.

486のFPU命令は386と同じで，追加や変更はありません．CPU内のレジスタでは，FPUに関係するビットで変更されたものがあります.

E.1.3　マルチプロセッサ支援機能

複数のCPUを搭載した場合，CPU間の連絡調整を支援するための機能です.

命令では，XADD(① XCHGと② ADDの処理をひとつの命令で実行)と，CMPXCHG(① CMP ＋ ② XCHGの処理をひとつの命令で実行)が，①と②の処理の間にほかのCPUのアクセスが入るのを排除できる連続実行命令として新設されています．これらを使えば，ほかのCPUから共有メモリに書かれたデータを参照して，そのことをほかのCPUに知らせるといったことが，タイミングのずれなく実現できます.

また，インテル系のCPUだけでなく，モトローラ系のCPUとのマルチプロセッサシステムにも容易に対応できるように，BSWAP(32ビットデータを8ビット単位で逆順にする)命令も追加されています．これを使えば，相手方のデータを自己のCPUの様式に合った形式で，または自己のデータを相手方のCPUの処理に合った様式で処理することができます.

E.1.4　486の追加命令一覧

キャッシュメモリとマルチプロセッサ支援機能のために，486で追加された命令は表E–1のとおりです.

▼表 E-1　80486 で追加された命令 (Mem はメモリ, Reg はレジスタ, R/M はレジスタまたはメモリ)

命令コード	ニーモニック	オペランド	機　能
0F01	INVLPG(invalidate TLB entry)	Mem	TLB エントリを削除する
0F08	INVD(invalidate data cache)		キャッシュ中のデータをすべて無効にする
0F09	WBINVD(write-back and invalidate data cache)		キャッシュ中のデータをすべて主記憶に書き出し，内容を無効にする
0FC0	XADD(exchange and add)	R/M, Reg	オペランドを 8 ビット単位で読み込み，レジスタ内容と加算して元に戻す
0FC1	XADD	R/M, Reg	オペランド内容を 16/32 ビット単位で読み込み，レジスタ内容と加算する
0FC8～0FCF	BSWAP(byte swap)	Reg	32 ビットレジスタ内容を 8 ビット単位でスワップ
0FA6	CMPXCHG(compare and exchange)	R/M, Reg	8 ビット単位でアキュムレータと第 1 オペランドを比較し，同じであれば第 2 オペランドが第 1 オペランドに入る
0FA7	CMPXCHG	R/M, Reg	16/32 ビット単位でアキュムレータと第 1 オペランドを比較し，同じであれば第 2 オペランドが第 1 オペランドに入る

E.1.5　486 内部レジスタのビット変更および追加

●拡張フラグレジスタ

▼図 E-2　拡張フラグレジスタ (EFLAGS) の追加ビット

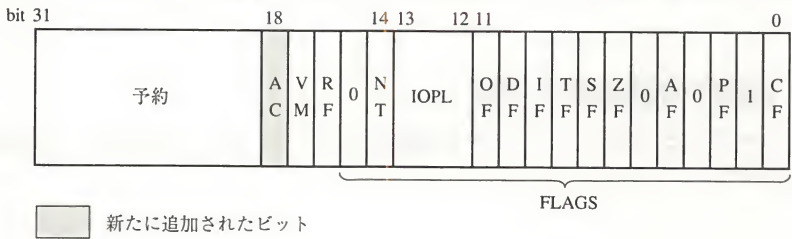



図 E–2 の拡張フラグレジスタ (EFLAGS) では，AC(アライメントチェック) ビットが追加されています．このビットは，0 のとき 386 互換ですが，1 にすると，特権レベル 3(ユーザモード) でアライメントチェックを行います．具体的には，データ属性のアライメント (ワード境界，ダブルワード境界) を調べ，それを無視したメモリ参照が発生すると，次に述べる CR0 の AM ビットが 1 の場合，アライメントフォルト (例外 17) の割り込みを起動します．

なお，アライメントチェックは，386 では行われておらず，単独 CPU のシステムでは問題になりませんが，マルチプロセッサのときはシビアにする必要があります．

●制御レジスタＣＲ０

制御レジスタ CR0(図 E–3) の追加および変更点は次のとおりです．

・CE(キャッシュイネーブル)

内蔵キャッシュを有効にする (1) か，無効にする (0) かを示します．有効のとき，ミスヒット (ヒットしない) が生じたら，外部のメモリから命令やデータを取り込みます．

・WT(透過書き込み)

内蔵キャッシュに書き込みが発生したとき，1 ならば外部のメモリの該当アドレスにも同じデータを書き込み，つじつまを合わせます (ライトスルー)．0 ならば，内蔵キャッシュには書き込みを行わず，外部のメモリにだけ書き込みします．

・AM(アライメントマスク)

このビットが 1 ならば，EFLAGS の AC ビットが 1(アライメントチェックを行う) で境界無視が検出されたとき，アライメントフォルトを発生します．0 のときは 386 互換となります．

・WP(ライトプロテクト)

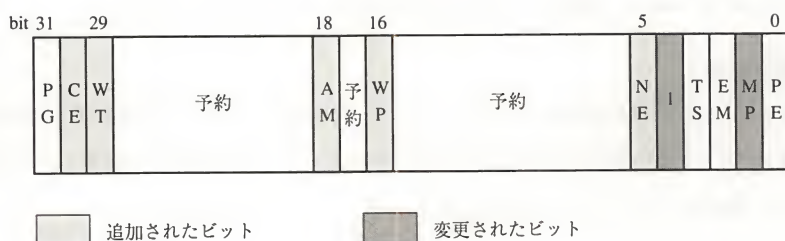
386 で可能だった，書き込み禁止ページへのスーパーバイザモードによる書き込みについて，486 ではこのビットを 1 にするとフォルトが発生し，防止することができます．WP＝0 のときは，386 互換となります．

・NE(数値例外)

FPU でマスクされていない例外が発生したとき，本来は例外 16 が適用されますが，このビットが 0 のときは，ハードウェアで便宜的に他のベクタに振り向けることができます．386 の時代によく行われていた手法で，それとの互換性を確保するためのものです．NE＝0 のときは，ベクタは 16 となります．

・MP(コプロセッサ表示)

386 では使われていた，FPU が外部に存在するかどうかを示すビット (1 のとき存在) は，ハードウェアの違いにより 486 では無意味になりました．386 との互換性のため残したもので，リセット時，0 になります．

▼図 E-3　制御レジスタ CR0 の追加/変更ビット

●制御レジスタＣＲ３

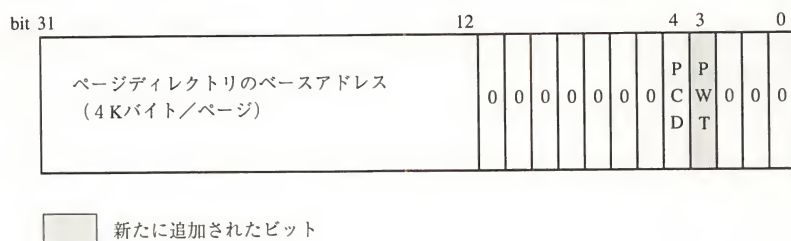
制御レジスタ CR3(図 E-4) で追加されたビットは次のとおりです.

・PWT(ページライトスルー)

外付けキャッシュメモリを使う場合，ライトスルーを行うならば1，ライトバック (キャッシュから元のメモリに書き戻す) 構造ならば0とします.

・PCD(ページキャッシュイネーブル)

ページディレクトリのベースアドレスで示されるページについて，内蔵キャッシュが有効 (0) か無効 (1) かを決めます．ただし，ハードウェアでキャッシュイネーブル (KEN) にアクティブ入力があるときのみ，有効にすることの意味があります．無効のときは，直接外部キャッシュまたはメモリに対してアクセスが行われます.

▼図 E-4　制御レジスタ CR3 の追加ビット

E.1.6　アドレスラップアラウンド

8086 プログラムでは，セグメントセレクタとオフセットの値の合計が FFFFH を超えるとき，10000H を差し引いた値，すなわちオーバーフローした値が採用され，アドレスラップアラウンド (先端と終端の連結) が行われていました.

80386では，このような場合，10000Hを超えるアドレスがアクセスされていましたが，80486ではラップアラウンドされるようになりました．これにより，8086との互換性が改善されています．

E.2　ソフトの不適合対策ヒント

先にも述べたように，486になっても386用のソフトのほとんどはそのまま使えます．ここでは，使えなくなるケースと，その場合の対策について触れておきます．

E.2.1　DOS-Extender〔RUN386〕で走るプログラムの場合

言い換えると，ネイティブモード(プロテクトモード)のプログラムで，386搭載機で使っていたDOS-Extenderをそのまま持ち込み，

RUN386 <プログラム名> <パラメータ> ……

とやっても，エラーメッセージもなく，何も実行されずに終了します．

これは，DOS-Extenderが486のレベルになっていないからで，486機システムソフトウェアCDからのものをコピーして，オーバーライトしてしまえば解決します．

念のために付け加えると，486用のDOS-Extenderも，プログラム名は〝RUN386〞のままです．すなわち，DOS-Extenderを呼び出すバッチファイル(〝～.BAT〞)の内容を変更する必要もないようになっているのです．

E.2.2　F-BASIC386の場合

386機で支障なく動いていたBASICプログラムが，486機に持って行って386機のままのF−BASIC386で動かすと，予想外のエラーメッセージが出て実行できなくなります．これは，486機ではF−BASIC386 V2.1のレベルでないと動作できないためで，レベルアップすれば解決します．

付録 F

FMTOWNSの製品系列

FMTOWNS 初代機が開発されてからの機種の変遷を，図 F−1 に示します．

80386 搭載の 2 機種からスタートした FMTOWNS は，ハードディスクが内蔵できるタイプ (1F,2F,1H,2H) に発展した後，規模を拡大 (10F,20F,40H,80H) して，FMTOWNS II へと展開しました．

TOWNS II になると，それまでの機種の発展型 (CX10,CX20,CX40,CX100) のほかに，さらに省スペースで低価格のディスプレイ一体型 (UX10,UX20,UX40) が加わりました．

続いて，セパレート型の機種のデザインが横型に変わり，80386 搭載機種 (HG20,HG40,HG100) と 80486 搭載機種 (HR20,HR100,HR200)，一体型は 80386 搭載機種 (UG10,UG20,UG40,UG80) の，それぞれパフォーマンスが強化された機種が発表されました．

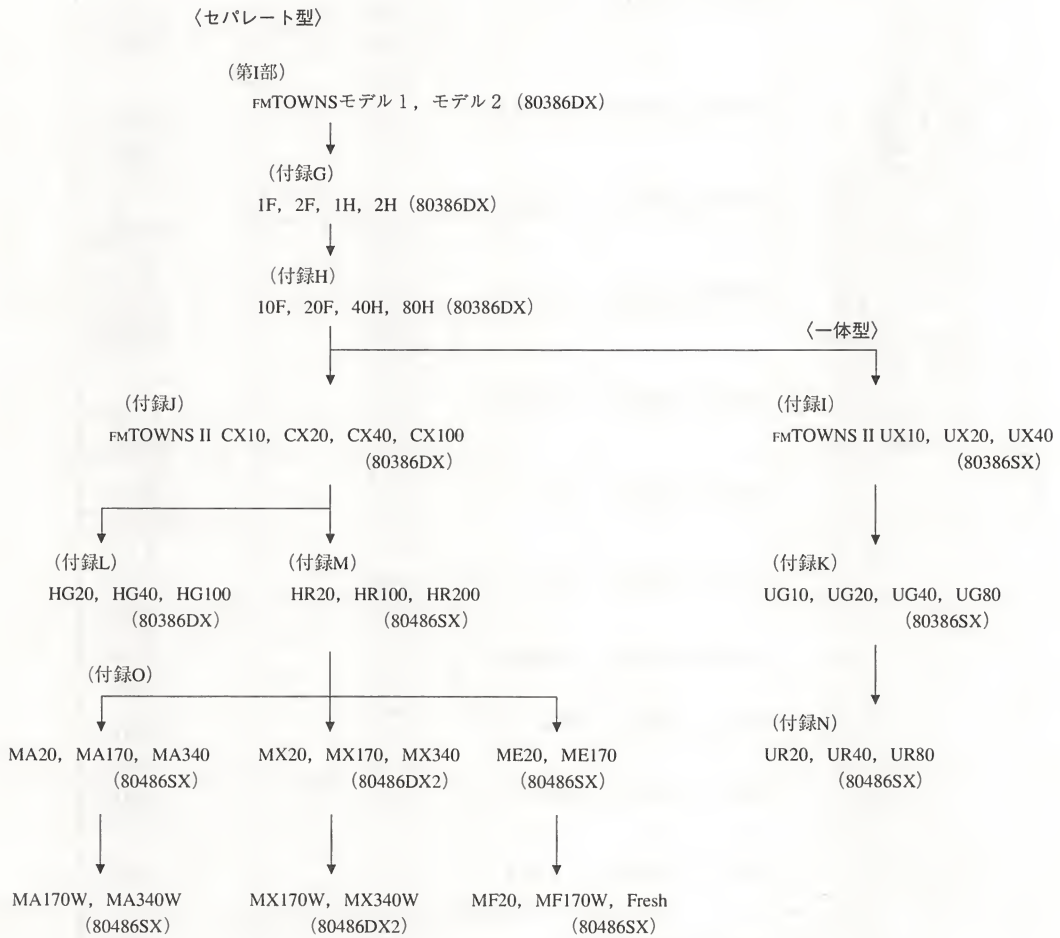
その後，さらに一体型にも 80486 搭載機 (UR20,UR40,UR80) ができ，これですべて 80486 への布陣が揃いました．FMTOWNS MARTY ができたのも，この時期です．

そして，以後の機種は CD−ROM を倍速化して展開します．Windows などの内外の規格に標準装備のまま対応できることを意識して高解像度化を図った機種では，CPU クロックのランクにより高速機 (MX20,MX170,MX340) と中速機 (MA20,MA170, MA340) が発表されました．また，同時に，入門者向けで，簡単な拡張により高解像度に対応できる機種 (ME20,ME170) も発表されました．

これらは，まもなく Windows3.1 インストール機 (MA170W,MA340W,MX170W,MX340W) へと展開し，ME の CPU ランク (クロック) を上げた機種 (MF20,MF170W,Fresh) もできました．

図 F−1 は，機種の変遷を示すと同時に，目的の機種の付録との関連も示しています，例えば，FMTOWNS II MX の場合，初代機→1F→10F→CX→HR→MX のように改良されているので，このようにさかのぼって関連記事を参照するのが適当です．

これらの機種の主な仕様の違いを，表 F−1 に示します．

▼図 F-1　FMTOWNS 初代機からの機種の変遷

▼表 F-1　FMTOWNS の主な仕様 (初代機以降の拡張内容)

機種名	CPU	FPU/ODP	メイン RAM	補助記憶 (ディスク)	
				3.5インチ2HD	ハードディスク
FMTOWNS モデル 1	80386DX(16MHz)	387 カード	1MB	1	なし
FMTOWNS モデル 2			2MB	2	なし
FMTOWNS 1F	80386DX(16MHz)	387 カード	1MB	1	なし
FMTOWNS 2F			2MB	2	なし
FMTOWNS 1H			1MB	2	20MB
FMTOWNS 2H			2MB	2	40MB
FMTOWNS 10F	80386DX(16MHz)	387 カード	2MB	1	なし
FMTOWNS 20F			2MB	2	なし
FMTOWNS 40H			2MB	2	40MB
FMTOWNS 80H			2MB	2	80MB
FMTOWNS II UX10	80386SX(16MHz)	接続不可	2MB	1	なし
FMTOWNS II UX20			2MB	2	なし
FMTOWNS II UX40			2MB	2	40MB
FMTOWNS II CX10	80386DX(16MHz)	387 カード	2MB	1	なし
FMTOWNS II CX20			2MB	2	なし
FMTOWNS II CX40			2MB	2	40MB
FMTOWNS II CX100			2MB	2	100MB
FMTOWNS II UG10	80386SX(20MHz)	接続不可	2MB	1	なし
FMTOWNS II UG20			2MB	2	なし
FMTOWNS II UG40			2MB	2	40MB
FMTOWNS II UG80			2MB	2	80MB
FMTOWNS II HG20	80386DX(20MHz)	387 カード	2MB	2	なし
FMTOWNS II HG40			2MB	2	40MB
FMTOWNS II HG100			2MB	2	100MB
FMTOWNS II HR20	80486SX(20MHz)	ODP カード	4MB	2	なし
FMTOWNS II HR100			4MB	2	100MB
FMTOWNS II HR200			4MB	2	200MB
FMTOWNS II UR20	80486SX(20MHz)	接続不可	2MB	2	なし
FMTOWNS II UR40			2MB	2	40MB
FMTOWNS II UR80			2MB	2	80MB
FMTOWNS II ME20	80486SX(25MHz)	ODP カード	2MB	2	なし
FMTOWNS II ME170			2MB	2	170MB
FMTOWNS II MA20	80486SX(33MHz)	ODP カード	4MB	2	なし
FMTOWNS II MA170			4MB	2	170MB
FMTOWNS II MA340			4MB	2	340MB
FMTOWNS II MX20	80486DX2(66MHz)	内蔵	4MB	2	なし
FMTOWNS II MX170			4MB	2	170MB
FMTOWNS II MX340			4MB	2	340MB
FMTOWNS II MF20	80486SX(33MHz)	ODP カード	4MB	2	なし
FMTOWNS II MF170W			6MB	2	170MB
FMTOWNS II Fresh*			6MB	2	170MB
FMTOWNS II MA170W	80486SX(33MHz)	ODP カード	8MB	2	170MB
FMTOWNS II MA340W			8MB	2	340MB
FMTOWNS II MX170W	80486DX2(66MHz)	内蔵	8MB	2	170MB
FMTOWNS II MX340W			8MB	2	340MB

*Fresh(OASYS/Win インストール済み) は MF170W と同じ.

付録G

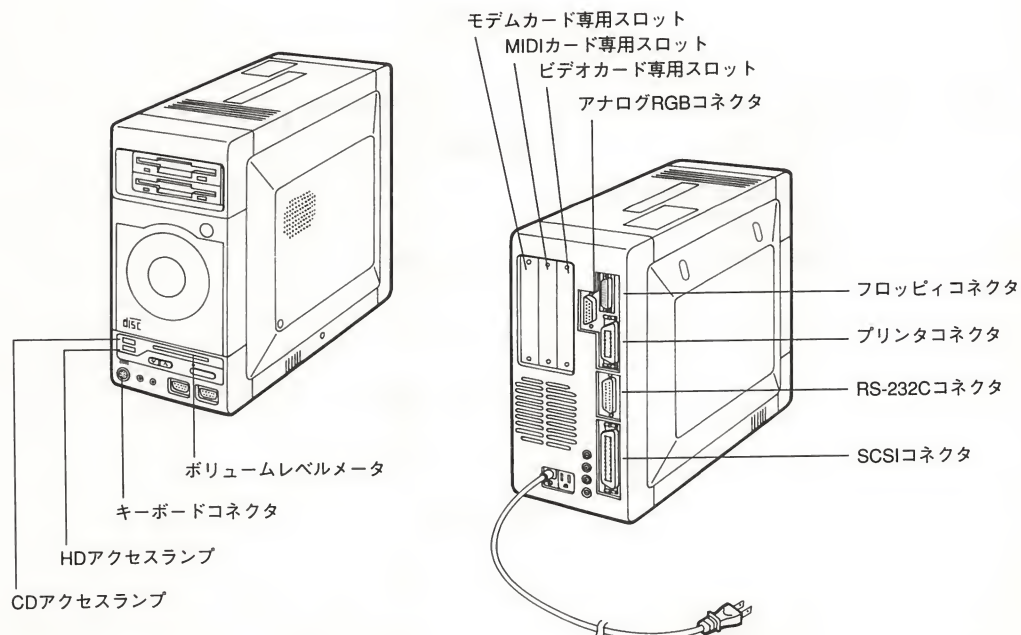
FMTOWNS 1F, 2F, 1H, 2Hの仕様変更

FMTOWNS 1F，2F，1H，2H の主な仕様変更部分について解説します．

機種名	メモリ	ハードディスク
1F	1MB	なし
2F	2MB	なし
1H	1MB	20MB
2H	2MB	40MB

以下に，仕様が変更された部分を解説します．

●外観

次の図のように外観が変更されています．

外観上の大きな変更点は，CD，HD のアクセスランプが新設されたこと，LED の数とデザインが変更されたこと，キーボードコネクタの位置が前面になったこと，SCSI コネクタが追加されたことなどです．また，本体背面のスロットの構成が変わりました．

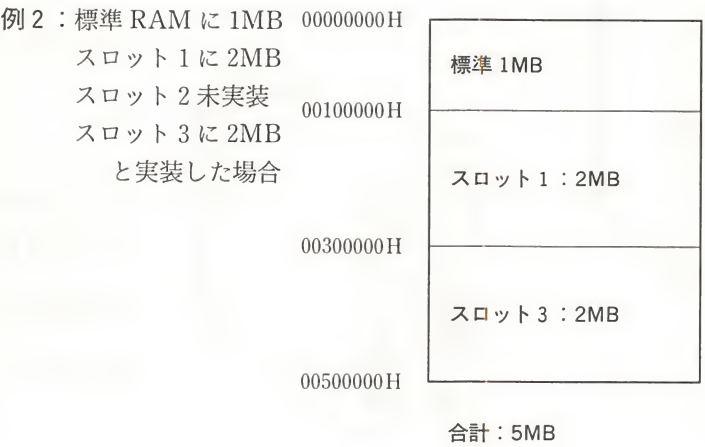
●拡張 RAM の内容

3 スロットある拡張 RAM モジュールコネクタにそれぞれ 1MB あるいは，2MB の RAM モジュールを実装することが可能です．実装する RAM モジュールの組み合せは自由で，メモリ空間が飛び飛びになることはありません．1MB 内蔵の機種で最大 7MB まで，2MB 内蔵の機種で最大 8MB まで拡張が可能です．

RAM 拡張の例を示します．

例 2 ：標準 RAM に 1MB
スロット 1 に 2MB
スロット 2 未実装
スロット 3 に 2MB
と実装した場合

00000000H
標準 1MB
00100000H
スロット 1 ：2MB
00300000H
スロット 3 ：2MB
00500000H

合計：5MB

●追加，および拡張されたI/Oとレジスタ

1．ドライブステータスレジスタ

フロッピィディスクの状態を示すレジスタです．FDCがMB89312に変更されたことに伴ない，ビット7，6，4，3，2，0の機能が拡張されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0208H	ドライブステータスレジスタ	R	FD2	0	不定	FDDV2	FDDV1	FDDV0	FREADY	DSK CHG

FD2(bit7)　：内蔵FDのドライブ数を示す．
0＝1ドライブ
1＝2ドライブ

FDDV2～FDDV0 (bit4～2)　：拡張ドライブの種別を示す．

FDDV2	FDDV1	FDDV0	ドライブの種別	bit0
0	0	0	予約済	
0	0	1	予約済	
0	1	0	予約済	
0	1	1	予約済	
1	0	0	予約済	
1	0	1	FMFD-322	DSKCHG
1	1	0	FMFD-521/801	DSK2S
1	1	1	FMFD-321	DSKCHG

FREADY(bit1)　：選択されたドライブの状態を示す．
0＝ノットレディ
1＝レディ

DSKCHG(bit0) (DSK2S)　：選択されたドライブにディスクチェンジがあったことを示す．
0＝チェンジあり
1＝チェンジなし
ドライブの種別がFMFD-521/801の場合は，DSK2Sであり，ディスクが両面か片面かを示す．

FREADYとDSKCHGの選択されたドライブとは，ドライブセレクトレジスタ(020CH)のDSL3-0ビットで選択されているものです．

2．ドライブコントロールレジスタ

ドライブを制御するためのレジスタです．ビット6の機能が拡張されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0208H	ドライブコントロールレジスタ	W	0	MOTR2	CLK SEL	MOTOR	0	HD1 SEL	DDEN	IRQ MSK

MOTR2 (bit6)　：外付け5インチのモータ動作を行う．
　0＝モータオフ
　1＝モータオン

CLKSEL (bit5)　：FDC に与えるクロックを指定する．
　0＝2MHz(3.5インチ，5インチ 2HD)
　1＝1MHz(3.5インチ，5インチ 2D/2DD)
シーク動作はこのビットを0としておくことにより，高速シークが可能である．
リード/ライト時は制御対象となるメディアに応じた設定(上記のとおり)としなければならない．

MOTOR (bit4)　：3.5インチ，5インチドライブのモータを制御する．
　0＝OFF(停止)
　1＝ON(回転)

HD1SEL (bit2)　：リード/ライトの対象となるメディアの面を指定する．
　0＝サイド0
　1＝サイド1

DDEN (bit1)　：メディアの記録方式を指定する．
　0＝単密度
　1＝倍密度

IRQMSK (bit0)　：FDC からの割り込みを制御する．
　0＝割り込み禁止
　1＝割り込み許可

3．メモリ容量レジスタ

メモリの容量を示すレジスタです．新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05E8H	メモリ容量レジスタ	R	不定	0		SIZE4	SIZE3	SIZE2	SIZE1	SIZE0

SIZE4-0 (bit4-0)　：メモリの容量を示す．ビットコードで0～32MB まで表される．

4．CPU 識別レジスタ

マシンの機種と CPU の種類を示すレジスタです．ID15-3 に新機種の機種 ID が加わりました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU 識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID (bit15-3)　：装置の種別を示す．次のビット構成により識別を行う．

装　置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FM TOWNS (1, 2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FM TOWNS (1F, 2F, 1H, 2H)	0	0	0	0	0	0	1	0	0	0	0	0	0

FM TOWNS 各機種の MACHINE-ID は，ID15-8 を使用し，ID7-3 が 0 のとき有効である．

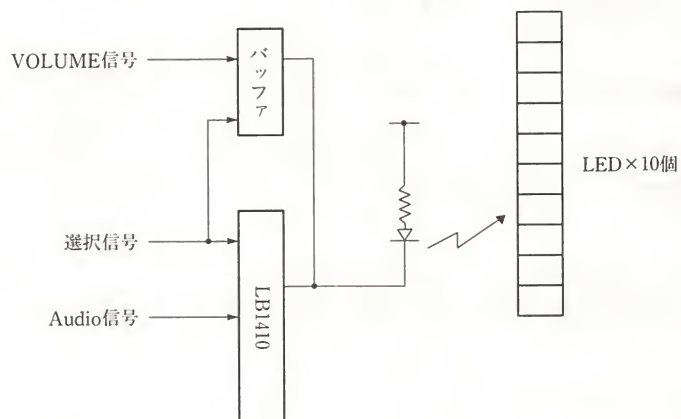
CPU-ID (bit2-0)　：使用 CPU の種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386
0	1	0	予約済
0	1	1	予約済
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

このレジスタ(ID15-ID0)で読み出される値(0201H)は，シリアル ROM 制御レジスタ(0032H)により読み出すことのできる機種番号(ビット 56-71)の値と等価です．

● LED

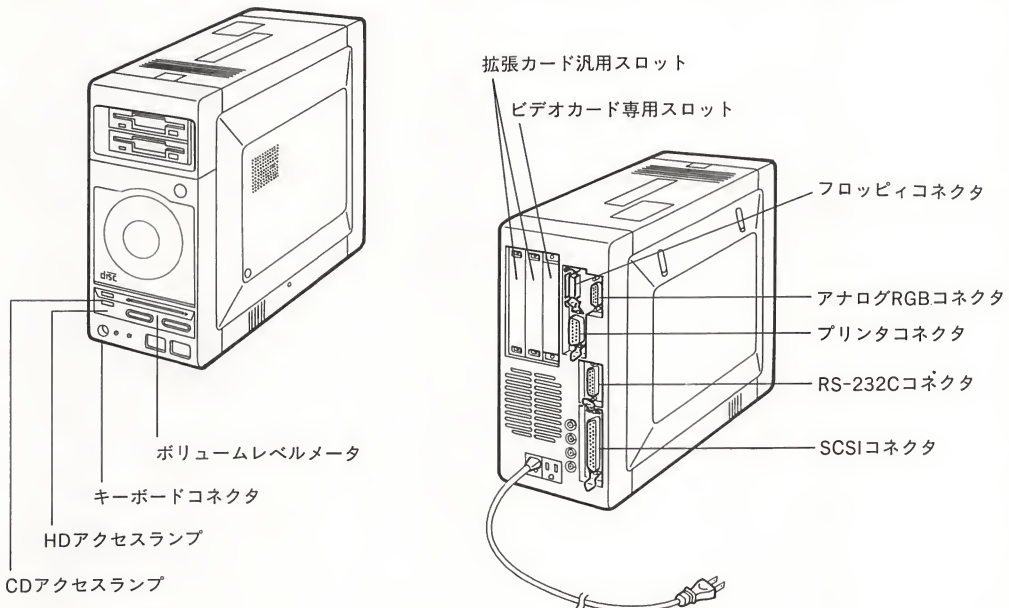
LED が 10 個に増えたことにともない，LED 関係のブロック構成は，次のように変更されました．

付録 H

FM TOWNS 10F, 20F, 40H, 80Hの仕様変更

FM TOWNS 10F, 20F, 40H, 80H の主な仕様変更部分について解説します.
なお,「付録G　FM TOWNS 1F, 2F, 1H, 2H の仕様変更」もあわせてお読みください.

機種名	メモリ	ハードディスク
10F	2MB	なし
20F	2MB	なし
40H	2MB	40MB
80H	2MB	85MB

●外観

次の図のように外観が変更されています.

本体背面の3つの拡張スロットの構成が変更になりました. 2つが拡張カード汎用スロット, 残り1つがビデオカード専用スロットとなっています.
内部的には, I/O拡張ユニットを接続するためのコネクタがなくなりました.

●追加, および拡張されたI/Oとレジスタ

1. 1μWAITレジスタ

1マイクロ秒のウェイトを, ハードウェアで作り出すレジスタです. このレジスタに書き込みを行うと, 1マイクロ秒のウェイトがかかります.

ソフトウェアのループによるウェイトでは, 機種の違いや, 同一機種でもメモリの性能が異なると時間に差が出ることがありますが, このレジスタを使用すれば, そういった問題はなくなります.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
006CH	1μWAITレジスタ	R	0	不定						
		W	0	0	0	0	0	0	0	0

2. インターバルタイマII関連レジスタ

インターバルタイマIIは, 主として音源制御に適したタイマで, PITを利用したインターバルタイマよりも細かな単位で時間間隔を設定したいときに使用します.

インターバルタイマII制御レジスタ

インターバルタイマIIの機能設定(書き込み)や，割り込み発生時の状態の参照(読み出し)を行うためのレジスタです．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0068H	インターバルタイマII 制御レジスタ	R	INTV -EN	INTV -I	INTV -OV	不定				
		W		0	0	0	0	0	0	0

INTV-EN(bit7)　：インターバルタイマIIによる割り込み許可．
　0＝許可する
　1＝許可しない
INTV-I(bit6)　：インターバルタイマIIによる割り込み発生の有無．
　0＝割り込み発生なし
　1＝割り込み発生
INTV-OV(bit5)　：以前のインターバルタイマIIによる割り込みが処理されないうちのインターバルタイマIIの再割り込みの発生の有無．
　0＝割り込み発生なし
　1＝割り込み発生

INTV-I, INTV-0は，このレジスタを読み出した直後，自動的に0になる．

インターバルタイマIIデータレジスタ

インターバルタイマIIの時間間隔を，マイクロ秒単位で設定するレジスタです．16ビットで，1～65535を直接表現します．0にすると65536として扱われます．

なお，タイマは1マイクロ秒のクロックで動作しているので，その範囲内で誤差が発生することがあります．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
006AH (下位)	インターバルタイマII データレジスタ	R/W	INTV -7	INTV -6	INTV -5	INTV -4	INTV -3	INTV -2	INTV -1	INTV -0
006BH (上位)		R/W	INTV -15	INTV -14	INTV -13	INTV -12	INTV -11	INTV -10	INTV -9	INTV -8

INTV-n　：設定値(マイクロ秒)を2進数16桁で表す．

3. CPU 識別レジスタ

マシンの機種と CPU の種類を示すレジスタです．ID15-3 に新機種の機種 ID が加わりました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU 識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID (bit15-3)　：装置の種別を示す．次のビット構成により識別を行う．

装　置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FM TOWNS (1, 2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FM TOWNS (1F, 2F, 1H, 2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FM TOWNS (10F, 20F, 40H, 80H)	0	0	0	0	0	1	0	0	0	0	0	0	0

FM TOWNS 各機種の MACHINE-ID は，ID15-8 を使用し，ID7-3 が 0 のとき有効である．

CPU-ID (bit2-0)　：使用 CPU の種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386
0	1	0	予約済
0	1	1	予約済
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

このレジスタ(ID15-ID0)で読み出される値(0401H)は，シリアル ROM 制御レジスタ(0032H)により読み出すことのできる機種番号(ビット 56-71)の値と等価です．

付録 I

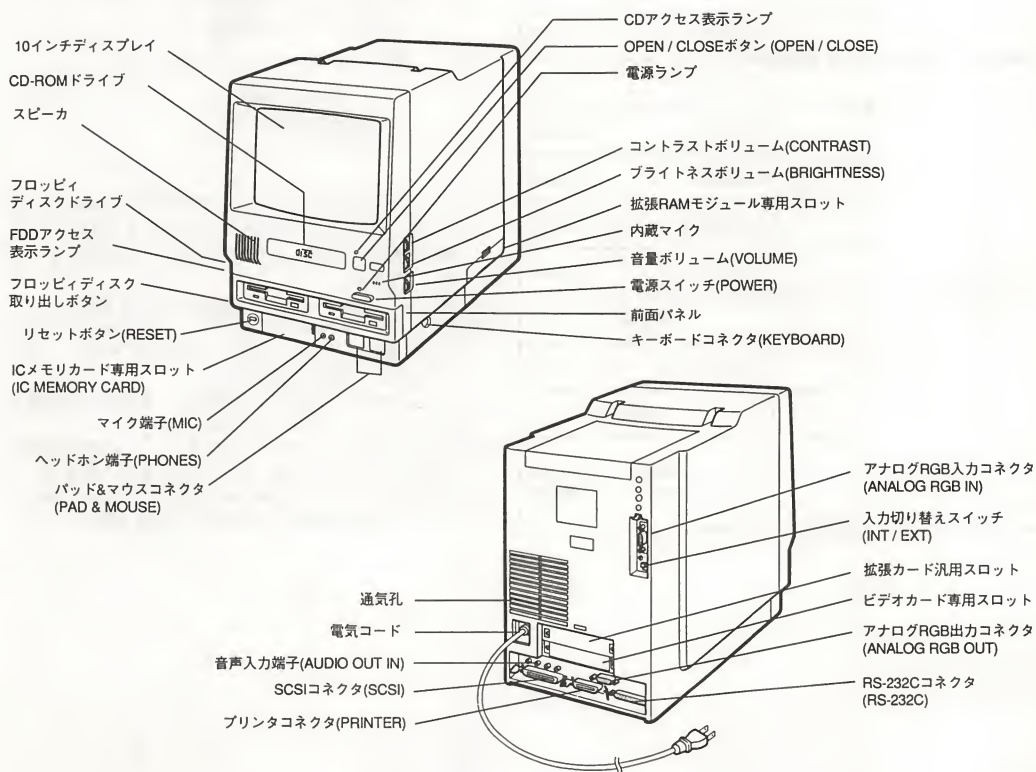
FMTOWNS II UXの仕様変更

FMTOWNS II UX の主な仕様変更部分について解説します．

なお，「付録G　FMTOWNS 1F，2F，1H，2H の仕様変更」，「付録H　FMTOWNS 10F，20F，40H，80H の仕様変更」もあわせてお読みください．

機種名	メモリ	FDD	ハードディスク
UX10	2MB	1	なし
UX20	2MB	2	なし
UX40	2MB	2	40MB

●外観

次の図のように外観が変更されています．

本体とディスプレイが一体型になりました．ディスプレイのみを利用できるように，アナログ RGB 入力端子が拡張されています．また，拡張カード汎用スロットが1つになりました．

●メモリマップ

CPU が 80386SX になったことにより，次の図のように変更がありました．

＊1 ブートROM領域は，FF8000-FFFFFFにあるが，リセット時，およびメモリ切り換えレジスタ(I/Oアドレス0480H番地)の操作により，0F8000-0FFFFFにもマッピングされる．
＊2 8MB RAMモジュール1枚実装時のメモリ容量．
＊3 VRAMは，2ヶ所マップされているが，表示とアクセス方法の違いによって，別アドレスに割り付けているだけである．実装は512KBのみ．
＊4 学習RAMの後の6KBは，外字フォントとしても使用される．バイトアクセスのみ可能．
＊5 波形RAMはバンク(16)レジスタにより64KBの領域を4KB単位で使用される．バイトアクセスのみ可能．
＊6 I/O拡張スロットのC80000-CFFFFFの領域は，汎用スロット上で40000000-4007FFFFに変換される．

● DMAC使用上の注意

FMTOWNS II UXでは，A24〜A31は80386(386DX)と互換をとるためアドレス変換を行っています．したがって，DMACを使う場合には，他機種とのアドレスの違いを意識する必要はありません．

変換しているアドレス

80000000H 〜 800FFFFFH	→	A00000H 〜 AFFFFFH	VRAM
80100000H 〜 801FFFFFH	→	B00000H 〜 BFFFFFH	VRAM
81000000H 〜 8107FFFFH	→	C00000H 〜 C7FFFFH	パターンRAM
C2200000H 〜 C2200FFFH	→	F80000H 〜 F80FFFH	PCMRAM
40000000H 〜 4007FFFFH	→	C80000H 〜 CFFFFFH	I/O拡張領域
C0000000H 〜 C0FFFFFFH	→	D00000H 〜 DFFFFFH	メモリカード
C2000000H 〜 C207FFFFH	→	E00000H 〜 E7FFFFH	OS-ROM
C2080000H 〜 C20FFFFFH	→	E80000H 〜 EFFFFFH	辞書ROM
C2100000H 〜 C213FFFFH	→	F00000H 〜 F3FFFFH	漢字ROM
C2140000H 〜 C2141FFFH	→	F40000H 〜 F41FFFH	学習RAM
FFFC0000H 〜 FFFFFFFFH	→	FC0000H 〜 FFFFFFH	システムROM

メモリカード領域は1MBのバンクになります．

●追加，および拡張されたI/Oとレジスタ

1. リセット要因レジスタ

リセットが発生したときに，その原因を示すレジスタです．ビット2の機能が拡張されました．

ソフトリセット，NMIベクタプロテクト，ソフト電源制御レジスタ(0020H)および電源制御レジスタ(0022H)を利用したソフトウェアによる電源のオフが起こった場合はCPUリセットがかかりビット2が1となります．POFFによるソフトリセットの場合は0404H，0480Hの各ビットをリセット(0に)します．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0020H	リセット要因レジスタ	R	不　定					POFF	SHUT DOWN	SOFT

POFF(bit2)　：ソフトウェアによるパワーオフリセットが発生したことを示す．このビットはリードしてもオフにならない．
　1＝パワーオフリセット

SHUTDOWN(bit1)　：シャットダウン(CPUによる異常検出)によるリセットが発生したことを示す．このビットはリードすることによりオフにされる．
　1＝シャットダウンリセット

SOFT(bit0)　：ソフトウェアによるリセットが発生したことを示す．このビットはリードすることによりオフにされる．
　1＝ソフトウェアリセット

シャットダウンリセット，ソフトウェアリセットともにCPUとNDPにのみリセットがかかる．
パワーオンリセットおよびシステムリセット時はいずれのフラグも0になる．

2. CPU識別レジスタ

マシンの機種とCPUの種類を示すレジスタです．ID15-3に新機種の機種IDが加わりました．

また，CPU識別レジスタのID0にメインメモリ領域のMEMINHを有効にするための書き込みビットができました．

MEMINHとは，本体内部にあるメモリ領域の一部を無効にして，汎用バスに実装されるオプションカードのメモリに置き換えるための信号です．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
		W	0	0	0	0	0	0	0	LSPEED
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID(bit15-3)　：装置の種別を示す．下記のビット構成により識別を行う．

装　置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不　定								1	1	1	1	1
FMR-50S	不　定								1	1	1	0	1
FMR-70	不　定								1	1	1	1	0
FMTOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FMTOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FMTOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FMTOWNS II (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FMTOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0

FMTOWNS各機種のMACHINE-IDは，ID15-8を使用し，ID7-3が0のとき有効である．

CPU-ID(bit2-0)　：使用CPUの種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386
0	1	0	80486
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

LSPEED(bit0)　：メインメモリ領域のMEMINHを有効にする．
0＝無効
1＝有効(このとき，RAMのウエイトは2WAITとなる)

このレジスタ(ID15-ID0)で読み出される値(0303H)は，シリアルROM制御レジスタ(0032H)により読み出すことのできる機種番号(ビット56-71)の値と等価です．

3. メモリカードバンクレジスタ

メモリカード領域を割り当てるレジスタです．JEIDA VER.4 対応のため，このレジスタとメモリカード属性レジスタが新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0490H	メモリカードバンクレジスタ	R/W	0	0	JB5	JB4	JB3	JB2	JB1	JB0

JB5-0(bit5-0)　：メモリカード領域(D00000H～E00000H)の 1MB をメモリカードの 64MB のどの領域に割り当てるかを指定します．

JB5	JB4	JB3	JB2	JB1	JB0	メモリ領域	
0	0	0	0	0	0	0MB～1MB	リセット時
0	0	0	0	0	1	1MB～2MB	
0	0	0	0	1	0	2MB～3MB	
0	0	0	0	1	1	3MB～4MB	
0	0	0	1	0	0	4MB～5MB	
0	0	0	1	0	1	5MB～6MB	
0	0	0	1	1	0	6MB～7MB	
0	0	0	1	1	1	7MB～8MB	
0	0	1	0	0	0	8MB～9MB	
0	0	1	0	0	1	9MB～10MB	
⋮	⋮	⋮	⋮	⋮	⋮	⋮	
1	1	1	0	1	1	59MB～60MB	
1	1	1	1	0	0	60MB～61MB	
1	1	1	1	0	1	61MB～62MB	
1	1	1	1	1	0	62MB～63MB	
1	1	1	1	1	1	63MB～64MB	

メモリカードのメモリをアクセスする際には，メモリカード属性レジスタ(0491H)の REG が 0 である必要があります．

4. メモリカード属性レジスタ

メモリカードの属性を読み取るレジスタです．新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0491H	メモリカード属性レジスタ	R	VER4	不　定						REG
		W	0	0	0	0	0	0	0	

VER4(bit7) ：サポートするメモリカードが JEIDA Ver.4 規格のものであることを示す．
0＝Ver.4 サポート
1＝Ver.4 未サポート
このビットが1のときメモリカードバンクレジスタとメモリカード属性レジスタは無効となる．

REG(bit0) ：メモリカードの属性情報をアクセスする．
0＝メモリをアクセス(リセット時)
1＝属性情報をアクセス

5. ステータスレジスタ

SCSI インタフェースの状態を示すためのレジスタです．ビット2の機能が拡張されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C32H	ステータスレジスタ	R	REQ	I/O	MSG	C/D	BUSY	EX (0)	INT	PERR

REQ(bit7) ：Information Transfer フェーズにおける転送要求を示す．
0＝転送要求なし
1＝転送要求あり

I/O(bit6) ：データの入出力の方向を示す．
0＝出力
1＝入力

MSG(bit5) ：データレジスタの内容がメッセージであるか，データであるかを示す．
0＝データ
1＝メッセージ

C/D(bit4) ：データレジスタの内容がコントロール情報であるか，データであるかを示す．
0＝データ
1＝コントロール情報(コマンド，ステータス，メッセージ)

BUSY(bit3) ：SCSI バスの状態を示す．
0＝解放されている
1＝使用中である

EX(bit2) ：コントロールレジスタの RMSK(bit5)の割り込み制御機能の有無を示す．
0＝割り込み機能が有効(FMTOWNS II UX では0)
1＝割り込み機能が無効

INT(bit1) ：割り込みが発生したことを示す．
0＝割り込みなし
1＝割り込みあり
この割り込みは，Command，Status，Message のいずれかのフェーズに移行し，REQ 信号が1になったときに発生する．ただし，コントロールレジスタの IMSK(bit6)に0を書くことによりマスクすることができる．

PERR(bit0) ：周辺装置からのデータのパリティエラーを示す．
0＝パリティエラーなし
1＝パリティエラーあり
ステータスレジスタを読むと，このビットは0になる．

6. コントロールレジスタ

SCSI インタフェースを制御するレジスタです．ビット 5，3 の情報が拡張されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C32H	コントロールレジスタ	W	WEN	IMSK	RMSK (0)	ATN	W/B (0)	SEL	DMAE	RST

WEN (bit7)　：SCSI バスへのデータ，コントロール信号の出力を制御する．
　0＝出力を禁止する
　1＝出力を許可する

IMSK (bit6)　：ステータスレジスタの INT (bit1) の割り込みを制御する．
　0＝割り込み禁止
　1＝割り込み許可

RMSK (bit5)　：データフェーズにおける割り込みの制御を行う．データフェーズ割り込みにおいては，IMSK は意味を持たない．
　0＝割り込み禁止 (FM TOWNS II UX では 0 に固定である)
　1＝割り込み許可

ATN (bit4)　：周辺装置に対して何らかのメッセージがあることを示す．
　0＝メッセージなし
　1＝メッセージあり

W/B (bit3)　：データフェーズ時の DMA 転送モードの設定を行う．
　0＝DMA 転送モードをバイト DMA モードにする (FM TOWNS II UX では 0 に固定である)
　1＝DMA 転送モードをワード DMA モードにする
　本ビットに 1 を指定したときは，DMA のチャネル 1 をワード転送に設定すること．

SEL (bit2)　：SCSI バスの SEL 信号の制御を行う．
　0＝OFF
　1＝ON

DMAE (bit1)　：DMA 転送を制御する．
　0＝DMA 転送を禁止する
　1＝DMA 転送を行う

RST (bit0)　：SCSI コネクタに接続しているすべての周辺装置をリセットする．
　0＝リセット解除
　1＝リセット (25μs 以上後に 0 にもどすこと)

7. CPU_MISC4 レジスタ

NMI のマスク機能の有無を示すレジスタです．新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0025H	CPU_MISC4 レジスタ	R	NMI CINT	1	1	1	1	1	1	1

NMICINT (bit7)　：NMI のマスク機能の有無を示す．
　0＝NMI のマスク機能あり
　1＝NMI のマスク機能なし

8. NMI マスクレジスタ

NMI のマスク機能の状態を変更するレジスタです．新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0028H	NMI マスクレジスタ	R/W	書き込み時には読み出した値を書き込むこと							NMI MASK

NMIMASK(bit0)　：NMI の禁止／有効を指定する．
0＝NMI が有効である
1＝NMI が禁止である（ただし，NMI 要因がクリアされるわけではないので，要因がある場合には，ビット 0 を 0 にすることによって NMI が有効となる）
リセット時，ビット 0 の値は 0 である．ソフトウェアリセット時でも状態は変化しない．
bit1〜7 のデータについては，読み出された値を書き込むこと．

9. BUFFUL レジスタ

シリアルデータが送信可能かどうかを示すレジスタです．新規に追加されました．

このレジスタはチューナカード(FMT-416)を実装したときに使用されるレジスタであり，チューナカードが実装されていない場合は何の意味も持ちません．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0606H	BUFFUL レジスタ	R	1	1	1	1	1	1	1	BUFFUL

BUFFUL(bit0)　：シリアルデータが送信可能かどうかを示す．
0＝送信可能
1＝送信不可能

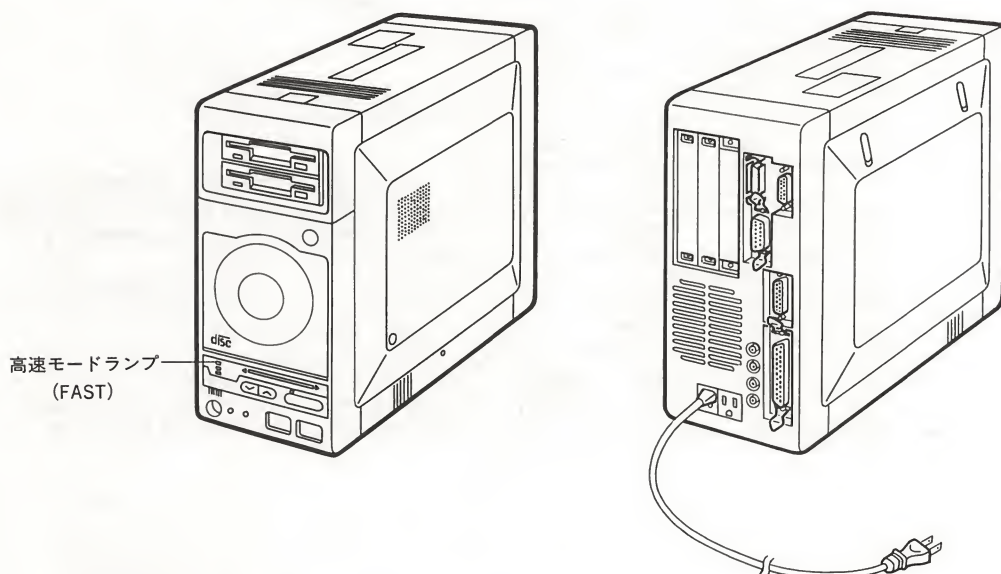
付録J

FMTOWNS II CXの仕様変更

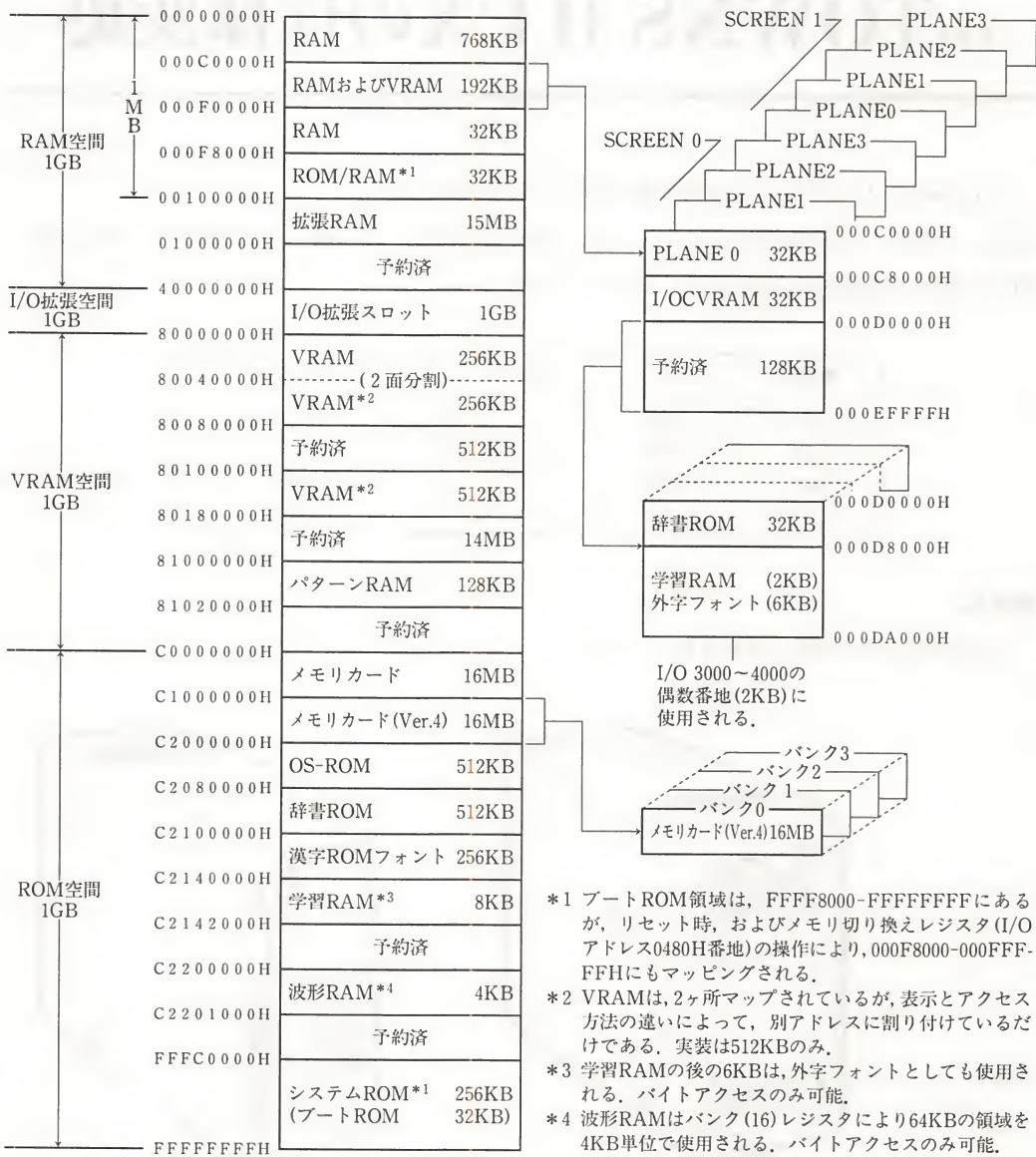
FMTOWNS II CX の主な仕様変更部分について解説します．
なお，「付録G FMTOWNS 1F，2F，1H，2H の仕様変更」「付録H FMTOWNS 10F，20F，40H，80H の仕様変更」もあわせてお読みください．

機種名	メモリ	FDD	ハードディスク
CX10	2MB	1	なし
CX20	2MB	2	なし
CX40	2MB	2	40MB
CX100	2MB	2	100MB

●外観

次の図のように外観が変更されています．

高速モードでの動作を示す，ランプが付きました．

●メモリマップ

メモリカード JEIDA Ver.4 対応のため，新たにメモリカード領域 16MB が追加され，その領域がバンク形式になり 64MB のメモリカードのアクセスが可能になりました．従来までのメモリカード領域も存在し，新メモリカード領域のバンク 0 と同じ内容になります．

＊1 ブートROM領域は，FFFF8000-FFFFFFFFにあるが，リセット時，およびメモリ切り換えレジスタ(I/Oアドレス0480H番地)の操作により，000F8000-000FFFFFHにもマッピングされる．

＊2 VRAMは，2ヶ所マップされているが，表示とアクセス方法の違いによって，別アドレスに割り付けているだけである．実装は512KBのみ．

＊3 学習RAMの後の6KBは，外字フォントとしても使用される．バイトアクセスのみ可能．

＊4 波形RAMはバンク(16)レジスタにより64KBの領域を4KB単位で使用される．バイトアクセスのみ可能．

●追加，および拡張されたI/Oとレジスタ

1. CPU識別レジスタ

マシンの機種とCPUの種類を示すレジスタです．ID15-3に新規種の機種IDが加わりました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID(bit15-3) ：装置の種別を示す．下記のビット構成により識別を行う．

装　置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不　定								1	1	1	1	1
FMR-50S	不　定								1	1	1	0	1
FMR-70	不　定								1	1	1	1	0
FMTOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FMTOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FMTOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FMTOWNS II (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FMTOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0

FMTOWNS各機種のMACHINE-IDは，ID15-8を使用し，ID7-3が0のとき有効である．

CPU-ID(bit2-0) ：使用CPUの種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386
0	1	0	80486
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

このレジスタ(ID15-ID0)で読み出される値(0501H)は，シリアルROM制御レジスタ(0032H)により読み出すことのできる機種番号(ビット56-71)の値と等価です．

2. スピード制御レジスタ

CPU の処理速度を切り替えるためのレジスタで，新規に追加されました．互換モードでは従来機種と同一の処理速度となります．高速モードでは，メイン RAM 0WAIT，VRAM 3WAIT，SCSI の DMA 転送サイクルの短縮(2μs／サイクル→ 1.65μs／サイクル)となり，本体の高速モードランプが点灯します．

スピード制御レジスタの bit7 はスピード制御機能の有無を示します．スピード制御機能が有効の場合は 0 となります．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05ECH	スピード制御レジスタ	R	HSPDEN (0)	不定						HI SPEED
		W	0	0	0	0	0	0	0	

HSPEED(bit0)　　：互換モード/高速モードを選択する．
0 ＝ 互換モード
1 ＝ 高速モード

HSPDEN(bit7)　　：スピード制御機能の有無を示す．
0 ＝ スピード制御機能が有効
1 ＝ スピード制御機能が無効

3. メモリカードバンクレジスタ

メモリカード領域を割り当てるレジスタです．JEIDA VER.4 対応のため，このレジスタとメモリカード属性レジスタが新設されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0490H	メモリカードバンクレジスタ	R	不　定		JB1	JB0	不　定			
		W	0	0			0	0	0	0

JB1,0(bit5，4)　　：メモリカード領域(C1000000H〜C1FFFFFFH)の 16MB 空間をメモリカードの 64MB のどの領域に割り当てるかを指定します．

JB1	JB0	メモリ領域	
0	0	0MB〜16MB	リセット時
0	1	16MB〜32MB	
1	0	32MB〜48MB	
1	1	48MB〜64MB	

メモリカードのメモリをアクセスする際には，メモリカード属性レジスタ(0491H)の REG が 0 である必要があります．

4. メモリカード属性レジスタ

メモリカードの属性を読み取るレジスタです．新規に追加されました．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0491H</td><td rowspan="2">メモリカード属性レジスタ</td><td>R</td><td>VER4</td><td colspan="6">不　定</td><td rowspan="2">REG</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

VER4(bit7)　：サポートするメモリカードがJEIDA Ver.4規格のものであることを示す.
　0＝Ver.4サポート
　1＝Ver.4未サポート
　このビットが1のときメモリカードバンクレジスタとメモリカード属性レジスタは無効となる.

REG(bit0)　：メモリカードの属性情報をアクセスする.
　0＝メモリをアクセス(リセット時)
　1＝属性情報をアクセス

5. ステータスレジスタ

SCSIインタフェースの状態を示すためのレジスタです．ビット2の機能が拡張されました.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C32H	ステータスレジスタ	R	REQ	I/O	MSG	C/D	BUSY	EX (0)	INT	PERR

REQ(bit7)　：Information Transferフェーズにおける転送要求を示す.
　0＝転送要求なし
　1＝転送要求あり

I/O(bit6)　：データの入出力の方向を示す.
　0＝出力
　1＝入力

MSG(bit5)　：データレジスタの内容がメッセージであるか，データであるかを示す.
　0＝データ
　1＝メッセージ

C/D(bit4)　：データレジスタの内容がコントロール情報であるか，データであるかを示す.
　0＝データ
　1＝コントロール情報(コマンド，ステータス，メッセージ)

BUSY(bit3)　：SCSIバスの状態を示す.
　0＝解放されている
　1＝使用中である.

EX(bit2)　：コントロールレジスタのRMSK(bit5)の割り込み制御機能の有無を示す.
　0＝割り込み機能が有効(FMTOWNS II CXでは0)
　1＝割り込み機能が無効

INT(bit1)　：割り込みが発生したことを示す.
　0＝割り込みなし
　1＝割り込みあり
　この割り込みは，Command，Status，Messageのいずれかのフェーズに移行し，REQ信号が1になったときに発生する．ただし，コントロールレジスタのIMSK(bit6)に0を書くことによりマスクすることができる.

PERR(bit0)　：周辺装置からのデータのパリティエラーを示す.
　0＝パリティエラーなし
　1＝パリティエラーあり
　ステータスレジスタを読むと，このビットは0になる.

6. コントロールレジスタ

SCSI インタフェースを制御するレジスタです．ビット 5，3 の機能が拡張されました．
DMA の高速転送を行うためには，以下の設定が必要となります．

・SCSI コントローラの設定をワード DMA モードにする．
・DMAC のチャネル 1 の設定をワード転送にする．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0C32H	コントロールレジスタ	W	WEN	IMSK	RMSK	ATN	W/B	SEL	DMAE	RST

WEN (bit7)　：SCSI バスへのデータ，コントロール信号の出力を制御する．
　0＝出力を禁止する
　1＝出力を許可する
IMASK (bit6)　：ステータスレジスタの INT (bit1) の割り込みを制御する．
　0＝割り込み許可
　1＝割り込み禁止
RMSK (bit5)　：データフェーズにおける割り込みの制御を行う．データフェーズ割り込みにおいては，IMSK は意味を持たない．
　0＝割り込み禁止 (FMTOWNS II CX では 0 に固定である)
　1＝割り込み許可
ATN (bit4)　：周辺装置に対して何らかのメッセージがあることを示す．
　0＝メッセージなし
　1＝メッセージあり
W/B (bit3)　：データフェーズ時の DMA 転送モードの設定を行う．
　0＝DMA 転送モードをバイト DMA モードにする
　1＝DMA 転送モードをワード DMA モードにする
本ビットに 1 を指定したときは，DMA のチャネル 1 をワード転送に設定すること．
SEL (bit2)　：SCSI バスの SEL 信号の制御を行う．
　0＝OFF
　1＝ON
DMAE (bit1)　：DMA 転送を制御する．
　0＝DMA 転送を禁止する
　1＝DMA 転送を行う
RST (bit0)　：SCSI コネクタに接続しているすべての周辺装置をリセットする．
　0＝リセット解除
　1＝リセット (25μs 以上後に 0 にもどすこと)

7. CPU_MISC4 レジスタ

NMI のマスク機能の有無を示すレジスタです．新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0025H	CPU_MISC4 レジスタ	R	NMI CNT	1	1	1	1	1	1	1

NMICINT(bit7)　：NMI のマスク機能の有無を示す．
　0＝NMI のマスク機能あり
　1＝NMI のマスク機能なし

8. NMI マスクレジスタ

NMI のマスク機能の状態を変更するレジスタです．新規に追加されました．

I/O アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0028H	NMI マスクレジスタ	R/W	書き込み時には読み出した値を書き込むこと							NMI MASK

NMIMASK(bit0)　：NMI の禁止／有効を指定する．
　0＝NMI が有効である
　1＝NMI が禁止である（ただし，NMI 要因がクリアされるわけではないので，要因がある場合には，ビット 0 を 0 にすることによって NMI が有効となる）
リセット時，ビット 0 の値は 0 である．ソフトウェアリセット時でも状態は変化しない．
bit1～7 のデータについては読み出された値を書き込むこと．

付録K

FMTOWNS II UGの仕様変更

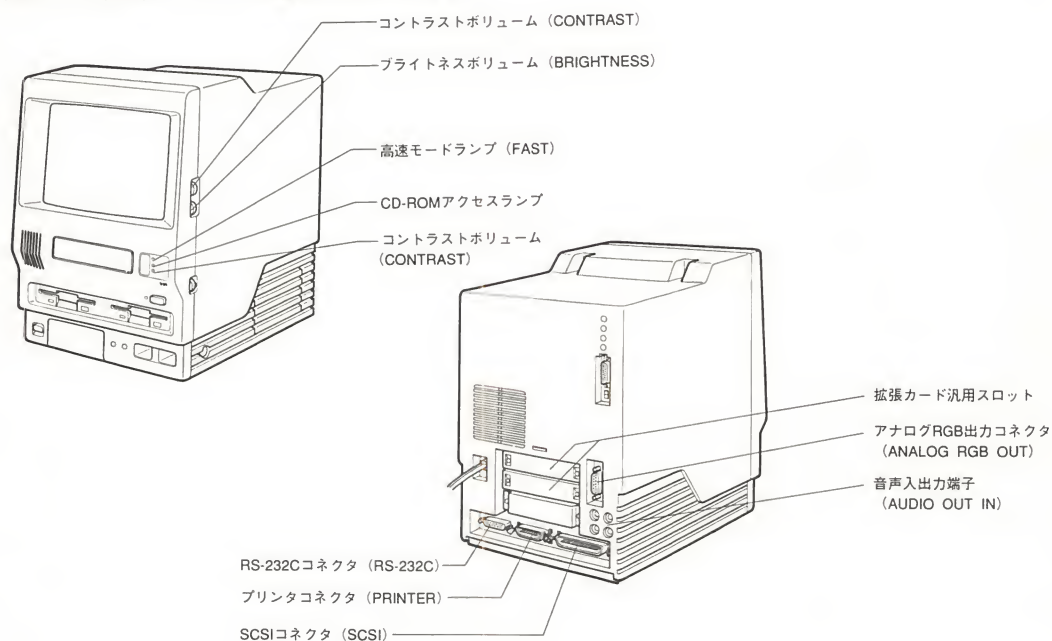
FMTOWNS II UG の主な仕様変更部分について解説します.

なお,「付録 G　FMTOWNS 1F,2F,1H,2H の仕様変更」,「付録 H　FMTOWNS 10F, 20F, 40H, 80H の仕様変更」,「付録 I　FMTOWNS II UX の仕様変更」もあわせてお読みください.

機種名	メモリ	FDD	ハードディスク
UG10	2MB	1	なし
UG20	2MB	2	なし
UG40	2MB	2	40MB
UG80	2MB	2	80MB

●外観

次の図のように外観が変更されています.

UX と比べて各種ボリュームやコネクタの位置が変わりました. また, 高速モードランプ, 拡張カード汎用スロットが増設されています.

●追加，および拡張されたI/Oとレジスタ

1. CPU識別レジスタ

CPU識別レジスタは，フォーマットの変更はありません．FMTOWNS II UGが追加されたことにより，図のビット構成(0603H)が参照されます．この値は，シリアルROM識別情報のビット56～71の内容と同じです．

なお，このレジスタのID15は，リセット直後は1となり，300μs後に0になります．0になったことを確認した上で参照してください．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID(bit15-3)　：装置の種別を示す．下記のビット構成により識別を行う．

装置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FM TOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FM TOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FM TOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FM TOWNS II (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FM TOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0
FM TOWNS II (UG10,UG20 UG40,UG80)	0	0	0	0	0	1	1	0	0	0	0	0	0

FMTOWNS各機種のMACHINE-IDは，ID15-8を使用し，ID7-3が0のとき有効である．

CPU-ID(bit2-0)　：使用CPUの種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386DX
0	1	0	80486SX/DX
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

2. CPU_MISC3 レジスタ

FMTOWNS II UG で新設されたレジスタ群が有効かどうかを参照するレジスタのひとつです．新規に追加されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0024H	CPU-MISC3レジスタ	R	ENPOFF (0)	RCREN (0)	CRT OFFEN	FTM (0)	POFFEN	1	1	1

ENPOFF(bit7)　：POFFEN(bit3) ビットが有効かどうかを示す．
　0 ＝ POFFEN ビット有効
　1 ＝ POFFEN ビット無効

RCREN(bit6)　：I/O FDA4H(リードコンパチレジスタ) が有効かどうか示す．
　0 ＝ リードコンパチレジスタが有効
　1 ＝ リードコンパチレジスタが無効

CRTOFFEN(bit5)　：I/O 0022H の CRTPOWOFF ビットが有効かどうかを示す．このビットはソフトウェアによる電源断可能な CRT が接続されていることを示している．
　0 ＝ CRT のソフト電源断が可能
　1 ＝ CRT のソフト電源断が不可能

FTM(bit4)　：フリーランタイマ (I/O 0026,0027H) の有無を示す．
　0 ＝ フリーランタイマ有
　1 ＝ フリーランタイマ無

POFFEN(bit3)　：I/O 0020H の POFF ビットが有効かどうかを示す．
　0 ＝ ソフト電源断が可能 (HR/HG)
　1 ＝ ソフト電源断が不可能 (UG/UR および従来機種)

3. 最高速クロックレジスタ

装置の最高動作周波数を表すレジスタです．新規に追加されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05EDH	最高速クロックレジスタ	R	FCLKEN (0)	FCLK6	FCLK5	FCLK4	FCLK3	FCLK2	FCLK1	FCLK0

FCLKEN(bit7)　：FCLK6-0 ビットが有効かどうかを示す．
　0 ＝ FCLK6-0 ビットが有効
　1 ＝ FCLK6-0 ビットが無効

FCLK6-0(bit6-0)　：装置の最高動作周波数を示す〔MHz〕
　0～127MHz

4. スピード制御レジスタ

CPU の処理速度を，従来機と同じ (互換モード) か，その装置の最高速にする (高速モード) かを選択するレジスタで，新規に追加されました．HSPDEN ビットは，0 (スピード制御機能あり) になっています．

最高速のときは，装置前面の高速モードランプが点灯します．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05ECH	スピード制御レジスタ	R	HSPDEN	不定						HSPEED
		W	0	0	0	0	0	0	0	

HSPDEN(bit7) ：スピード制御機能の有無を示す．
0 ＝ スピード制御機能が有効
1 ＝ スピード制御機能が無効

HSPEED(bit0) ：互換モード／高速モードを選択する (bit7＝0のとき有効).
0 ＝ 互換モード (386DX メイン RAM 3WAIT, V-RAM 6WAIT 相当)
1 ＝ 高速モード

5. フリーランタイマレジスタ

新規に追加されたレジスタです．このレジスタは，一種のタイマとして用いるためのもので，1μsごとに1ずつカウント値が増加します．このレジスタを使い，65ms以内の範囲で2回読み取って差を求めることにより，経過時間(μs単位)を算出することができます．ただし，内容は刻々変化するので，ワード読み出しで一度に読み取ることが必要です．もしバイト読み出しを2回行って上位と下位を連結すると，途中で値が変化して数値が不正確になることがあります．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0026H	フリーランタイマレジスタ	R	カウント値(下位)							
			CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
0027H		R	カウント値(上位)							
			CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8

注) このレジスタのリードに当たっては，必ずワード命令でリードすること．

6. SCSIモードステータスレジスタ

SCSIのDMAワード転送が可能かどうかを表すレジスタで，新規に追加されました．ただし，FMTOWNS II CXでは，ビット7が1(ワード転送不可)になっていますが，これは例外です．CXについては，CPU識別レジスタで機種を確認し，転送可能とみなしてください．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0034H	SCSIモードステータスレジスタ	R	WORD DMA	1	1	1	1	1	1	1

WORDDMA(bit7) ：DMAのワード転送が可能であるかどうかを示す．
0 ＝ DMAのワード転送可
1 ＝ DMAのワード転送不可

注) CXでは，DMAのワード転送が可能であるが，このビットは1になっているので，機種IDで判別すること．

7. CRT 出力コントロールレジスタ

新規に追加されたレジスタで，同名の書き込み専用レジスタ (I/O アドレス：FDA0H) の値を読み出すためのものです.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FDA2H	CRT出力コントロールレジスタ	R	0	0	0	0	画面レイア0		画面レイア1	
							COLOR	GREEN	COLOR	GREEN

注）このレジスタは，I/O FDA0H にライトした値をリードするためのものである.

8. CRT リードコンパチブルレジスタ

FMTOWNS II UG で新設されたレジスタ群が有効かどうかを参照するレジスタのひとつです. 具体的には，次に示すレジスタ群が存在するかどうか，言い換えるとほかの機種で動く可能性のあるソフトウェアの場合，これらが使えないことを知るために参照します.

このレジスタ自身も新設されたもので，これがないとき (従来機などの場合)，REN の値は 0 になります.

I/O アドレス：

FDA2H(CRT 出力コントロールレジスタ：Read)

メモリマップド I/O(メモリアドレス)：

000CFF88H(グラフィック VRAM ディスプレイモードレジスタ：Read)

000CFF99H(漢字 VRAM レジスタ：Read)

000CFF9CH(漢字 CG アクセスレジスタ 2 上位：Read)

000CFF9DH(漢字 CG アクセスレジスタ 2 下位：Read)

000CFF9EH(CG ROW アドレスレジスタ：Read/Write)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FDA4H	リードコンパチブルレジスタ	R	不定							REN
		W	0	0	0	0	0	0	0	

REN(bit0)　：I/O FDA2H，メモリマップド I/O FF88H,FF99H,FF9CH,FF9DH のリードおよびメモリマップド I/O FF9EH のリード／ライトが可能かどうかを示す.
このレジスタは，RCREN(I/O 0026H の bit6) が 0 のとき有効となる.
0 ＝ リード／ライト不可（従来互換，リセット時）
1 ＝ リード／ライト可能

9. グラフィック VRAM ディスプレイモードレジスタ

新規に追加されたレジスタで，同名の書き込み専用レジスタ (メモリマップド I/O メモリアドレス：000CFF82H) の値を読み出すためのものです.

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">000C
FF88H</td><td rowspan="2">グラフィックVRAM
ディスプレイモードレジスタ</td><td rowspan="2">R</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">RAM4</td><td colspan="2">PAGE SELECT</td><td colspan="3">RAM SELECT</td></tr>
<tr><td>PS2</td><td>0</td><td>RAM3</td><td>RAM2</td><td>RAM1</td></tr>
</table>

注）このレジスタは，メモリマップドI/O FF82Hにライトした値をリードするためのものである．

10. 漢字VRAMレジスタ

新規に追加されたレジスタで，同名の書き込み専用レジスタ（メモリマップドI/Oメモリアドレス：000CFF99H)の値を読み出すためのものです．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF99H	漢字VRAMレジスタ	R	0	0	0	0	0	0	0	ANKCG

注）このレジスタは，メモリマップドI/O FF99Hにライトした値をリードするためのものである．

11. 漢字CGアクセスレジスタ2

新規に追加されたレジスタで，同名の書き込み専用レジスタ（メモリマップドI/Oメモリアドレス：000CFF94H,000CFF95H)の値を読み出すためのものです．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>000C
FF9CH</td><td rowspan="2">漢字CGアクセスレジスタ2</td><td>R</td><td colspan="8">漢字コード(上位)</td></tr>
<tr><td>000C
FF9DH</td><td>R</td><td colspan="8">漢字コード(下位)</td></tr>
</table>

注）このレジスタは，メモリマップドI/O 000CFF94H, 000CFF95Hにライトした値をリードするためのものである．

12. CG ROWアドレスレジスタ

新規に追加されたレジスタで，漢字CGアクセスレジスタ（メモリマップドI/Oメモリアドレス：000CFF96H,000CFF97H)のRead/Writeデータ部分のROWアドレスを指定するためのものです．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">000C
FF9EH</td><td rowspan="2">CG ROWアドレスレジスタ</td><td>R</td><td colspan="4">不定</td><td rowspan="2">L3</td><td rowspan="2">L2</td><td rowspan="2">L1</td><td rowspan="2">L0</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

L3-0(bit3-0)　：メモリマップドI/O 000CFF96H, 000CFF97HのROWアドレスを指定する．

付録L

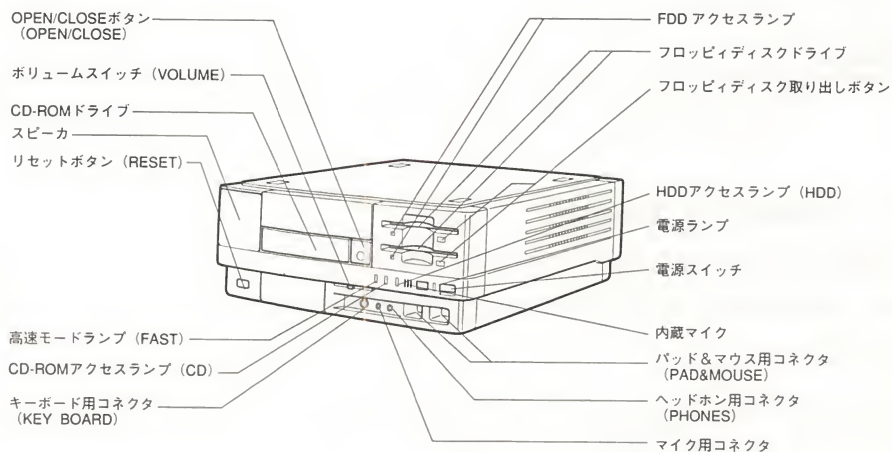
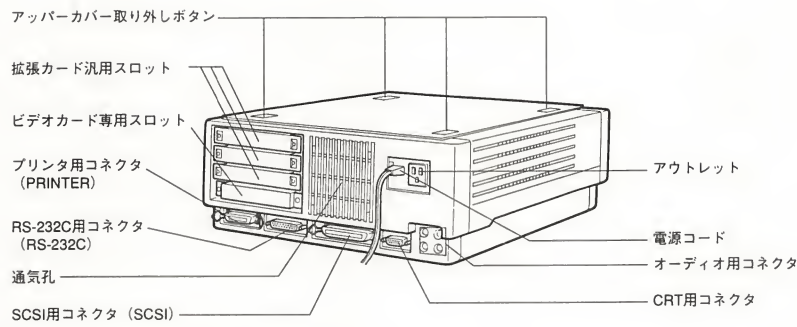
FM TOWNS II HGの仕様変更

FM TOWNS II HG の主な仕様変更部分について解説します.

なお,「付録G　FM TOWNS 1F,2F,1H,2H の仕様変更」,「付録H　FM TOWNS 10F, 20F, 40H, 80H の仕様変更」,「付録J　FM TOWNS II CX の仕様変更」もあわせてお読みください.

機種名	メモリ	FDD	ハードディスク
HG20	2MB	2	なし
HG40	2MB	2	40MB
HG100	2MB	2	100MB

●外観

次の図のように外観が横型に変更されています.

●追加，および拡張されたI/Oとレジスタ

1. CPU識別レジスタ

CPU識別レジスタは，フォーマットの変更はありません．FMTOWNS II HGが追加されたことにより，図のビット構成(0801H)が参照されます．この値は，シリアルROM識別情報のビット56～71の内容と同じです．

なお，このレジスタのID15は，リセット直後は1となり，300μs後に0になります．0になったことを確認した上で参照してください．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID(bit15-3)　：装置の種別を示す．下記のビット構成により識別を行う．

装置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FM TOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FM TOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FM TOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FM TOWNS II (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FM TOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0
FM TOWNS II (UG10,UG20 UG40,UG80)	0	0	0	0	0	1	1	0	0	0	0	0	0
FM TOWNS II (HG20,HG40 HG100)	0	0	0	0	1	0	0	0	0	0	0	0	0

FMTOWNS 各機種の MACHINE-ID は，ID15-8 を使用し，ID7-3 が 0 のとき有効である．

CPU-ID(bit2-0)　　：使用 CPU の種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386DX
0	1	0	80486SX/DX
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

2. 電源制御レジスタ

CRT の電源を，ソフトウェアで ON/OFF するためのビット (CRTPOWOFF) が追加されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0022H	電源制御レジスタ	R	CRT POWOFF	不定						
		W		POWOFF	0	0	0	0	0	0

CRTPOWOFF(bit7)：ソフトウェアにて CRT 電源の ON/OFF をする．
0 ＝ CRT 電源を ON にする
1 ＝ CRT 電源を OFF にする

3. CPU_MISC3 レジスタ

FMTOWNS II HG で新設されたレジスタ群が有効かどうかを参照するレジスタのひとつです．新規に追加されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0024H	CPU-MISC3レジスタ	R	ENPOFF (0)	RCREN (0)	CRT OFFEN	FTM (0)	POFFEN	1	1	1

ENPOFF(bit7)　：POFFEN(bit3) ビットが有効かどうかを示す.
0 ＝ POFFEN ビット有効
1 ＝ POFFEN ビット無効

RCREN(bit6)　：I/O FDA4H(リードコンパチレジスタ) が有効かどうか示す.
0 ＝ リードコンパチレジスタが有効
1 ＝ リードコンパチレジスタが無効

CRTOFFEN(bit5)　：I/O 0022H の CRTPOWOFF ビットが有効かどうかを示す. このビットはソフトウェアによる電源断可能な CRT が接続されていることを示している.
0 ＝ CRT のソフト電源断が可能
1 ＝ CRT のソフト電源断が不可能

FTM(bit4)　：フリーランタイマ (I/O 0026,0027H) の有無を示す.
0 ＝ フリーランタイマ有
1 ＝ フリーランタイマ無

POFFEN(bit3)　：I/O 0020H の POFF ビットが有効かどうかを示す.
0 ＝ ソフト電源断が可能 (HR/HG)
1 ＝ ソフト電源断が不可能 (UG/UR および従来機種)

4. 最高速クロックレジスタ

装置の最高動作周波数を表すレジスタです. 新規に追加されました.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05EDH	最高速クロックレジスタ	R	FCLKEN (0)	FCLK6	FCLK5	FCLK4	FCLK3	FCLK2	FCLK1	FCLK0

FCLKEN(bit7)　：FCLK6-0 ビットが有効かどうかを示す.
0 ＝ FCLK6-0 ビットが有効
1 ＝ FCLK6-0 ビットが無効

FCLK6-0(bit6-0)　：装置の最高動作周波数を示す〔MHz〕
0～127MHz

5. フリーランタイマレジスタ

新規に追加されたレジスタです. このレジスタは, 一種のタイマとして用いるためのもので, 1μs ごとに 1 ずつカウント値が増加します. このレジスタを使い, 65ms 以内の範囲で 2 回読み取って差を求めることにより, 経過時間 (μs 単位) を算出することができます. ただし, 内容は刻々変化するので, ワード読み出しで一度に読み取ることが必要です. もしバイト読み出しを 2 回行って上位と下位を連結すると, 途中で値が変化して数値が不正確になることがあります.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0026H	フリーランタイマレジスタ	R	カウント値(下位)							
			CNT7	CNT6	CNT5	CNT4	CNT3	CNT2	CNT1	CNT0
0027H		R	カウント値(上位)							
			CNT15	CNT14	CNT13	CNT12	CNT11	CNT10	CNT9	CNT8

注）このレジスタのリードに当たっては，必ずワード命令でリードすること．

6. FD ドライブステータスレジスタ

FD ドライブの種別を表すビット (FDDV2～0) が追加されました．3 モードドライブ対応の機能です．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0208H	FD ドライブステータスレジスタ	R	FD2	FDC	0	FDDV2	FDDV1	FDDV0	FREADY	DSKCHG

FDDV2-0(bit4-2)　：ドライブセレクトレジスタで指定したドライブの種別を示す．
UG/UR および従来機種では外付 FDD の種別を示す．
HR/HG では，ドライブ ＝0，＝1(内蔵ドライブ) を選択時には 011 となる．

FDDV2	FDDV1	FDDV0	ドライブ種別	bit0
0	0	0	予約済	—
0	0	1	予約済	—
0	1	0	予約済	—
0	1	1	3.5” 3 モード	DSKCHG
1	0	0	予約済	DSK2S
1	0	1	3.5” 2HD/2DD	DSKCHG
1	1	0	5” 2HD/2DD	DSK2S
1	1	1	3.5” 2HD	DSKCHG

7. FD ドライブセレクトレジスタ

従来からの HISPD ビットとの組み合わせでドライブの回転数を表す MODE-B ビットが追加されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
020CH	FD ドライブセレクトレジスタ	R	MODE-B	HISPD	0	INUSE	DSL3	DSL2	DSL1	DSL0

MODE-B(bit7)　：3.5”, 5”ドライブの回転数を指定する．

HISPD(bit6)

MODE-B	HISPD	回転数	備考
0	0	300rpm	2DD
0	1	360rpm	2HD
1	0	180rpm	2ED (未サポート)
1	1	300rpm	2HD 1.44MB

注) MODE-B(bit7), HISPD(bit6), INUSE(bit4)は，DSL-0(bit3-0)を1にしたときラッチされる．このため，MODE-B, HISPD, INUSEをセット／リセットしてからドライブ指定を行う（レジスタへの書き込みを二度行う）こと．

8. FD ドライブ識別レジスタ

FDドライブステータスレジスタで，FDドライブの種別を表すビット(FDDV2～0)が有効(識別できる)かどうかを表すレジスタです．FMTOWNS II HGでは0(有効)になっています．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
020DH	FDドライブ識別レジスタ	R	FDDV EXT	1	1	1	1	1	1	1

FDDVEXT(bit7)　：ドライブ識別ビット(FDDV2-0)で，内蔵ドライブの種別を識別できるように拡張されていることを示す．
0 = 拡張されており，内蔵ドライブの機番を指定することによって，FDDV2-0で内蔵ドライブの種別を識別できる(HR/HG)
1 = 拡張されておらず，内蔵ドライブの識別はできない(UG/URおよび従来機種)

9. SCSI モードステータスレジスタ

SCSIのDMAワード転送が可能かどうかを表すレジスタで，新規に追加されました．ただし，FMTOWNS II CXでは，ビット7が1(ワード転送不可)になっていますが，これは例外です．CXについては，CPU識別レジスタで機種を確認し，転送可能とみなしてください．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0034H	SCSIモードステータスレジスタ	R	WORD DMA	1	1	1	1	1	1	1

WORDDMA(bit7)　：DMAのワード転送が可能であるかどうかを示す．
0 = DMAのワード転送可
1 = DMAのワード転送不可

注) CXでは，DMAのワード転送が可能であるが，このビットは1になっているので，機種IDで判別すること．

10. CRT 出力コントロールレジスタ

新規に追加されたレジスタで，同名の書き込み専用レジスタ (I/O アドレス：FDA0H) の値を読み出すためのものです．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FDA2H	CRT出力コントロールレジスタ	R	0	0	0	0	画面レイア0		画面レイア1	
							COLOR	GREEN	COLOR	GREEN

注) このレジスタは，I/O FDA0H にライトした値をリードするためのものである．

11. CRT リードコンパチブルレジスタ

FMTOWNS II HG で新設されたレジスタ群が有効かどうかを参照するレジスタのひとつです．具体的には，次に示すレジスタ群が存在するかどうか，言い換えるとほかの機種で動く可能性のあるソフトウェアの場合，これらが使えないことを知るために参照します．

このレジスタ自身も新設されたもので，これがないとき (従来機などの場合)，REN の値は 0 になります．

I/O アドレス：

FDA2H(CRT 出力コントロールレジスタ：Read)

メモリマップド I/O(メモリアドレス)：

000CFF88H(グラフィック VRAM ディスプレイモードレジスタ：Read)

000CFF99H(漢字 VRAM レジスタ：Read)

000CFF9CH(漢字 CG アクセスレジスタ 2 上位：Read)

000CFF9DH(漢字 CG アクセスレジスタ 2 下位：Read)

000CFF9EH(CG ROW アドレスレジスタ：Read/Write)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
FDA4H	CRT リードコンパチブルレジスタ	R	不定							REN
		W	0	0	0	0	0	0	0	

REN(bit0) ：I/O FDA2H, メモリマップド I/O FF88H,FF99H,FF9CH,FF9DH のリードおよびメモリマップド I/O FF9EH のリード／ライトが可能かどうかを示す．
このレジスタは，RCREN(I/O 0026H の bit6) が 0 のとき有効となる．
0 ＝ リード／ライト不可 (従来互換，リセット時)
1 ＝ リード／ライト可能

12. グラフィックVRAMディスプレイモードレジスタ

新規に追加されたレジスタで，同名の書き込み専用レジスタ(メモリマップドI/Oメモリアドレス：000CFF82H)の値を読み出すためのものです．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">000C
FF88H</td><td rowspan="2">グラフィックVRAM
ディスプレイモードレジスタ</td><td rowspan="2">R</td><td rowspan="2">0</td><td rowspan="2">1</td><td rowspan="2">RAM4</td><td colspan="2">PAGE SELECT</td><td colspan="3">RAM SELECT</td></tr>
<tr><td>PS2</td><td>0</td><td>RAM3</td><td>RAM2</td><td>RAM1</td></tr>
</table>

注) このレジスタは，メモリマップドI/O FF82Hにライトした値をリードするためのものである．

13. 漢字VRAMレジスタ

新規に追加されたレジスタで，同名の書き込み専用レジスタ(メモリマップドI/Oメモリアドレス：000CFF99H)の値を読み出すためのものです．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
000C FF99H	漢字VRAMレジスタ	R	0	0	0	0	0	0	0	ANKCG

注) このレジスタは，メモリマップドI/O FF99Hにライトした値をリードするためのものである．

14. 漢字CGアクセスレジスタ2

新規に追加されたレジスタで，同名の書き込み専用レジスタ(メモリマップドI/Oメモリアドレス：000CFF94H,000CFF95H)の値を読み出すためのものです．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>000C
FF9CH</td><td rowspan="2">漢字CGアクセスレジスタ2</td><td>R</td><td colspan="8">漢字コード(上位)</td></tr>
<tr><td>000C
FF9DH</td><td>R</td><td colspan="8">漢字コード(下位)</td></tr>
</table>

注) このレジスタは，メモリマップドI/O 000C FF94H, 000C FF95Hにライトした値をリードするためのものである．

15. CG ROWアドレスレジスタ

新規に追加されたレジスタで，漢字CGアクセスレジスタ(メモリマップドI/Oメモリアドレス：000CFF96H,000CFF97H)のRead/Writeデータ部分のROWアドレスを指定するためのものです．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">000C
FF9EH</td><td rowspan="2">CG ROWアドレスレジスタ</td><td>R</td><td colspan="4">不定</td><td rowspan="2">L3</td><td rowspan="2">L2</td><td rowspan="2">L1</td><td rowspan="2">L0</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

L3-0(bit3-0)　：メモリマップドI/O 000CFF96H, 000CFF97HのROWアドレスを指定する．

付録 M

FM TOWNS II HRの仕様変更

FMTOWNS II HR の主な仕様変更部分について解説します.

なお,「付録 G FMTOWNS 1F,2F,1H,2H の仕様変更」,「付録 H FMTOWNS 10F, 20F, 40H, 80H の仕様変更」,「付録 J FMTOWNS II CX の仕様変更」,「付録 L FMTOWNS II HG の仕様変更」もあわせてお読みください.

機種名	メモリ	FDD	ハードディスク
HR20	4MB	2	なし
HR100	4MB	2	100MB
HR200	4MB	2	200MB

●外観

外観は HG と基本的に同じです.

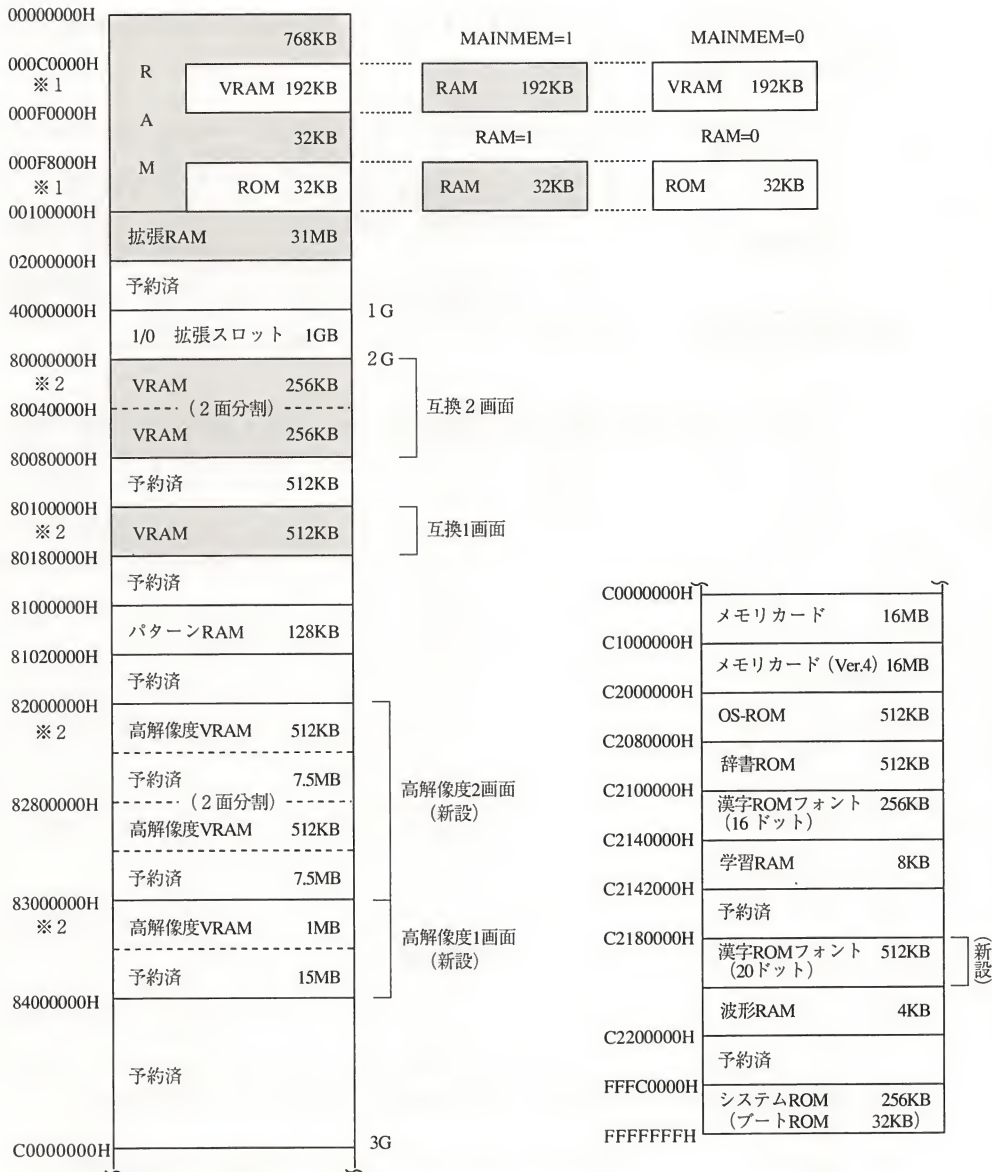
●メモリマップ

80486SX 採用のため,キャッシュメモリが使用可能になるなどの変更があります.図中の網かけ部分がキャッシュ使用可能領域です.

※1 000C0000～000EFFFF，000F8000～000FFFFFの領域は，裏RAMのみキャッシュ領域とする．
000C0000～000EFFFFの領域は，RAM→VRAMの切り替え時，000F8000～000FFFFFの領域はRAM→ROMの切り替え時にキャッシュをパージする．

※2 VRAM領域は，VCMENビット（I/O 05EEHのbit0）が1のときにのみキャッシュ領域となる．ただし，スプライト転送デジタイズおよびVIW時はキャッシュオフとなる．

●追加，および拡張されたI/Oとレジスタ

1. CPU識別レジスタ

CPU識別レジスタは，フォーマットの変更はありません．FMTOWNS II HRが追加されたことにより，図のビット構成(0702H)が参照されます．この値は，シリアルROM識別情報のビット56～71の内容と同じです．

なお，このレジスタのID15は，リセット直後は1となり，300μs後に0になります．0になったことを確認した上で参照してください．

I/0アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID(bit15-3)　：装置の種別を示す．下記のビット構成により識別を行う．

装置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FM TOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FM TOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FM TOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FM TOWNS II (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FM TOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0
FM TOWNS II (UG10,UG20 UG40,UG80)	0	0	0	0	0	1	1	0	0	0	0	0	0
FM TOWNS II (HG20,HG40 HG100)	0	0	0	0	1	0	0	0	0	0	0	0	0
FM TOWNS II (HR20,HR100 HR200)	0	0	0	0	0	1	1	1	0	0	0	0	0

FM TOWNS各機種のMACHINE-IDは，ID15-8を使用し，ID7-3が0のとき有効である．

CPU-ID(bit2-0)　：使用CPUの種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386DX
0	1	0	80486SX/DX
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

2. キャッシュ制御レジスタ

80486 内蔵キャッシュの動作を制御するレジスタで，新規に追加されました．高速モードのときのみ有効です．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00C0H	キャッシュ制御レジスタ	R	不定						RPINH	CMEN
		W	0	0	0	0	0	0		

RPINH(bit1)　：リプレース動作を禁止する (リセット時は 0).
　0 ＝ リプレース動作可能
　1 ＝ リプレース動作禁止

CMEN(bit0)　：キャッシュメモリを動作させる (リセット時は 0).
このビットは，HSPEED(I/O 05ECH の bit0) と連動してセット/リセットされる．
このビットを 1(キャッシュメモリ動作可) から 0(キャッシュメモリ動作禁止) にしたとき，キャッシュが全パージされる．
　0 ＝ キャッシュメモリ動作禁止
　1 ＝ キャッシュメモリ動作可

3. キャッシュ診断レジスタ

80486 内蔵キャッシュの診断に使われるレジスタで，新規に追加されました．高速モードのときのみ有効です．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00C2H	キャッシュ診断レジスタ	R	不定				SDMOD	不定		
		W	0	0	0	0		0	0	0

SDMOD(bit3)　：キャッシュメモリの診断に使用する．1 にすると，キャッシュメモリがデータ用のキャッシュとなり，コマンドは主記憶からリードするのみとなる．
　0＝キャッシュメモリ通常動作
　1＝キャッシュメモリ診断中

4. VRAM キャッシュ制御レジスタ

VRAM キャッシュの動作を制御するレジスタで，新規に追加されました．高速モードのときのみ有効です．VRAM のキャッシュ対象領域については，メモリマップを参照してください．

キャッシュのパージとともに，CPU 外部のメモリなどとのデータの整合性を保つため，I/O ビットについても次のように再設定が行われます．

全パージ条件	I/O アドレス	ビット名	ビット動作
キャッシュ動作禁止時	00C0H	CMEN	1 ⇒ 0
	05ECH	HSPEED	1 ⇒ 0
	05EEH	VCMEN	1 ⇒ 0
000C0000H ～ 000EFFFFH の領域を VRAM に切り替えたとき	0404H	MAINMEM	1 ⇒ 0
000F8000H ～ 000FFFFFH の領域を ROM に切り替えたとき	0480H	RAM	1 ⇒ 0
スプライト転送を開始したとき (VRAM キャッシュ有効時のみ)	0452H (01H)	SPEN	0 ⇒ 1
デジタイズを開始したとき (VRAM キャッシュ有効時のみ)	0442H (1CH)	ESYN ESM 0/1	1 0 ⇒ 1
他の VRAM をアクセスしたとき			
・VRAM1(2 画面)⇒ PLANE	—	—	—
・VRAM2(1 画面)⇒ PLANE	—	—	—
・VRAM1(2 画面)⇔ VRAM2(1 画面)	—	—	—

キャッシュ対象領域についてはメモリマップを参照のこと.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05EEH	VRAMキャッシュ制御レジスタ	R	0	不 定						VCMEN
		W	0	0	0	0	0	0	0	

(bit7)　：スピード制御機能の有無を示す.
　　0 = VRAM キャッシュ制御機能が有効
　　1 = VRAM キャッシュ制御機能が無効

VCMEN(bit0)　：VRAM 領域のキャッシュ制御を行う (bit7=0 のとき有効).
　このビットは, HSPEED(I/O 05ECH の bit0) と連動してセット/リセットされる.
　このビットを 1(VRAM キャッシュ動作可) から 0(VRAM キャッシュ動作禁止) にしたとき, キャッシュが全パージされる.
　　0 = VRAM 領域キャッシュに入れない (リセット時)
　　1 = VRAM 領域キャッシュに入れる

5. CD-ROM キャッシュ制御レジスタ

新規に追加されたレジスタです. CD-ROM 高速アクセスのとき使用されるキャッシュの制御を行います.

CD-ROM キャッシュの制御を行う. 本レジスタは高速モード時のみ有効となる.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04C8H	CD-ROMキャッシュ制御レジスタ	R	CACHE	不定						CACHE EN
		W	0	0	0	0	0	0	0	

CACHE(bit7)　：CD-ROM キャッシュ機能が搭載されていることを示す.
　　0 = CD-ROM キャッシュあり (HR)
　　0 = CD-ROM キャッシュなし (HG/UG および従来機種)

CACHEEN(bit0)　：CD-ROM キャッシュを有効にするかを選択する (bit7=0 のとき有効)
　このビットは, HSPEED(I/O 05ECH の bit0) と連動してセット／リセットされる.
　　0 = CD-ROM キャッシュを無効にする (リセット時)
　　1 = CD-ROM キャッシュを有効にする

注) CD-romXA モード時には, CD-ROM キャッシュは無効となる.

付録 N

FMTOWNSⅡ URの仕様変更

FMTOWNS II UR の主な仕様変更部分について解説します.

なお,「付録 G FMTOWNS 1F,2F,1H,2H の仕様変更」,「付録 H FMTOWNS 10F, 20F, 40H, 80H の仕様変更」,「付録 I FMTOWNS II UX の仕様変更」,「付録 K FMTOWNS II UG の仕様変更」もあわせてお読みください.

機種名	メモリ	FDD	ハードディスク
UR20	2MB	2	なし
UR40	2MB	2	40MB
UR80	2MB	2	80MB

●外観

外観は UG と基本的に同じです.

●メモリマップ

80486SX 採用のため，キャッシュメモリが使用可能になるなどの変更があります. 図中の網かけ部分がキャッシュ使用可能領域です.

●追加，および拡張された I/O とレジスタ

1. CPU 識別レジスタ

CPU 識別レジスタは，フォーマットの変更はありません．FMTOWNS II UR が追加されたことにより，図のビット構成 (0902H) が参照されます．この値は，シリアル ROM 識別情報のビット 56～71 の内容と同じです．

なお，このレジスタのID15は，リセット直後は1となり，300μs後に0になります．0になったことを確認した上で参照してください．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0030H	CPU識別レジスタ	R	MACHINE-ID					CPU-ID		
			ID7	ID6	ID5	ID4	ID3	ID2	ID1	ID0
0031H		R	MACHINE-ID							
			ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8

MACHINE-ID(bit15-3)　：装置の種別を示す．下記のビット構成により識別を行う．

装置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FM TOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FM TOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FM TOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FM TOWNS II (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FM TOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0
FM TOWNS II (UG10,UG20 UG40,UG80)	0	0	0	0	0	1	1	0	0	0	0	0	0
FM TOWNS II (HG20,HG40 HG100)	0	0	0	0	1	0	0	0	0	0	0	0	0
FM TOWNS II (HR20,HR100 HR200)	0	0	0	0	0	1	1	1	0	0	0	0	0
FM TOWNS II (UR20,UR40 UR80)	0	0	0	0	1	0	0	1	0	0	0	0	0

FMTOWNS各機種のMACHINE-IDは，ID15-8を使用し，ID7-3が0のとき有効である．

CPU-ID(bit2-0)　：使用CPUの種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386DX
0	1	0	80486SX/DX
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

2. キャッシュ制御レジスタ

80486 内蔵キャッシュの動作を制御するレジスタで，新規に追加されました．高速モードのときのみ有効です．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">00C0H</td><td rowspan="2">キャッシュ制御レジスタ</td><td>R</td><td colspan="6">不 定</td><td rowspan="2">RPINH</td><td rowspan="2">CMEN</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

RPINH(bit1)　：リプレース動作を禁止する (リセット時は 0).
　0 ＝ リプレース動作可能
　1 ＝ リプレース動作禁止

CMEM(bit0)　：キャッシュメモリを動作させる (リセット時は 0).
このビットは，HSPEED(I/O 05ECH の bit0) と連動してセット/リセットされる.
このビットを 1(キャッシュメモリ動作可) から 0(キャッシュメモリ動作禁止) にしたとき，キャッシュが全バージされる.
　0 ＝ キャッシュメモリ動作禁止
　1 ＝ キャッシュメモリ動作可

3. キャッシュ診断レジスタ

80486 内蔵キャッシュの診断に使われるレジスタで，新規に追加されました．高速モードのときのみ有効です．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">00C2H</td><td rowspan="2">キャッシュ診断レジスタ</td><td>R</td><td colspan="4">不 定</td><td rowspan="2">SDMOD</td><td colspan="3">不 定</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

SDMOD(bit3)　：キャッシュメモリの診断に使用する．1 にすると，キャッシュメモリがデータ用のキャッシュとなり，コマンドは主記憶からリードするのみとなる.
　0＝キャッシュメモリ通常動作
　1＝キャッシュメモリ診断中

4. VRAM キャッシュ制御レジスタ

VRAM キャッシュの動作を制御するレジスタで，新規に追加されました．高速モードのときのみ有効です．VRAM のキャッシュ対象領域については，メモリマップを参照してください．

キャッシュのパージとともに，CPU 外部のメモリなどとのデータの整合性を保つため，I/O ビットについても次のように再設定が行われます.

全パージ条件	I/O アドレス	ビット名	ビット動作
キャッシュ動作禁止時	00C0H	CMEN	1 ⇒ 0
	05ECH	HSPEED	1 ⇒ 0
	05EEH	VCMEN	1 ⇒ 0
000C0000H ～ 000EFFFFH の領域を VRAM に切り替えたとき	0404H	MAINMEM	1 ⇒ 0
000F8000H ～ 000FFFFFH の領域を ROM に切り替えたとき	0480H	RAM	1 ⇒ 0
スプライト転送を開始したとき (VRAM キャッシュ有効時のみ)	0452H (01H)	SPEN	0 ⇒ 1
ディジタイズを開始したとき (VRAM キャッシュ有効時のみ)	0442H (1CH)	ESYN ESM 0/1	1 0 ⇒ 1
他の VRAM をアクセスしたとき ・VRAM1(2 画面)⇒ PLANE ・VRAM2(1 画面)⇒ PLANE ・VRAM1(2 画面)⇔ VRAM2(1 画面)	 — — —	 — — —	 — — —

キャッシュ対象領域についてはメモリマップを参照のこと.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05EEH	VRAMキャッシュ制御レジスタ	R	0	不定						VCMEN
		W	0	0	0	0	0	0	0	

(bit7)　：スピード制御機能の有無を示す.
　0 ＝ VRAM キャッシュ制御機能が有効
　1 ＝ VRAM キャッシュ制御機能が無効

VCMEN(bit0)　：VRAM 領域のキャッシュ制御を行う (bit7＝0 のとき有効).
このビットは，HSPEED(I/O 05ECH の bit0) と連動してセット/リセットされる.
このビットを 1(VRAM キャッシュ動作可) から 0(VRAM キャッシュ動作禁止) にしたとき，キャッシュが全パージされる.
　0 ＝ VRAM 領域キャッシュに入れない (リセット時)
　1 ＝ VRAM 領域キャッシュに入れる

付録 O

FM TOWNS II ME,MA,MX,MF,Freshの仕様変更

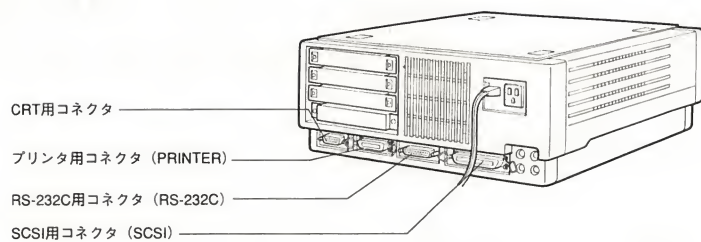
FM TOWNS II ME，MA，MX，MF の主な仕様変更部分について解説します．

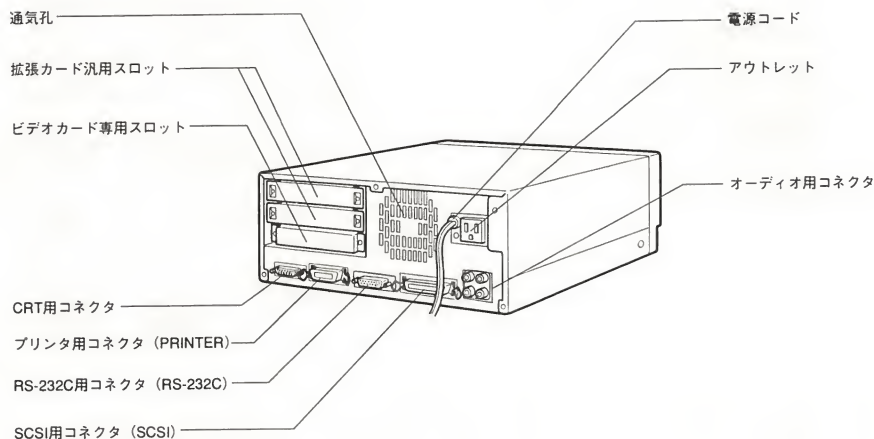
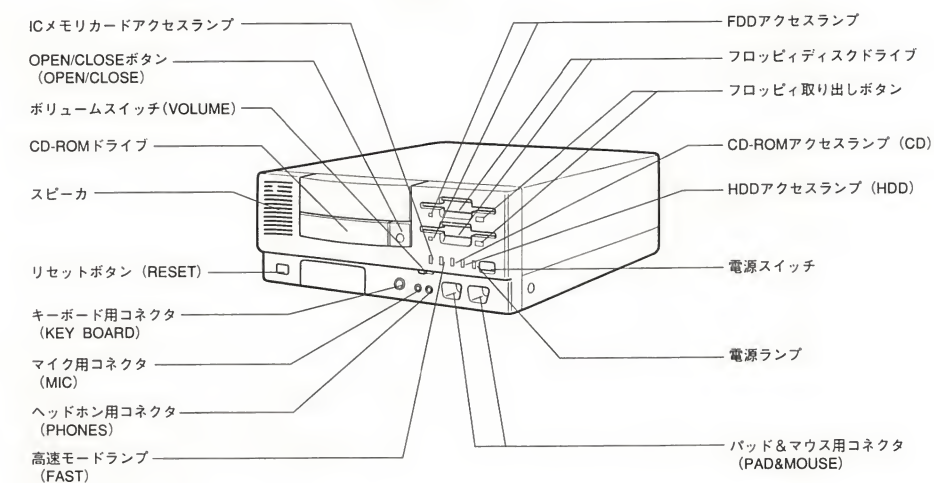
これらには，Windows インストール機 (機種名の末尾に〝W〟付き) および Fresh も含まれます．このうち，Fresh は MF170W に OASYS/Win をインストールしたもので，ハードウェアは MF に分類されます．

なお,「付録 G　FM TOWNS 1F,2F,1H,2H の仕様変更」,「付録 H　FM TOWNS 10F, 20F, 40F, 80F の仕様変更」,「付録 J　FM TOWNS II CX の仕様変更」,「付録 K　FM TOWNS II HG の仕様変更」,「付録 M　FM TOWNS II HR の仕様変更」もあわせてお読みください．

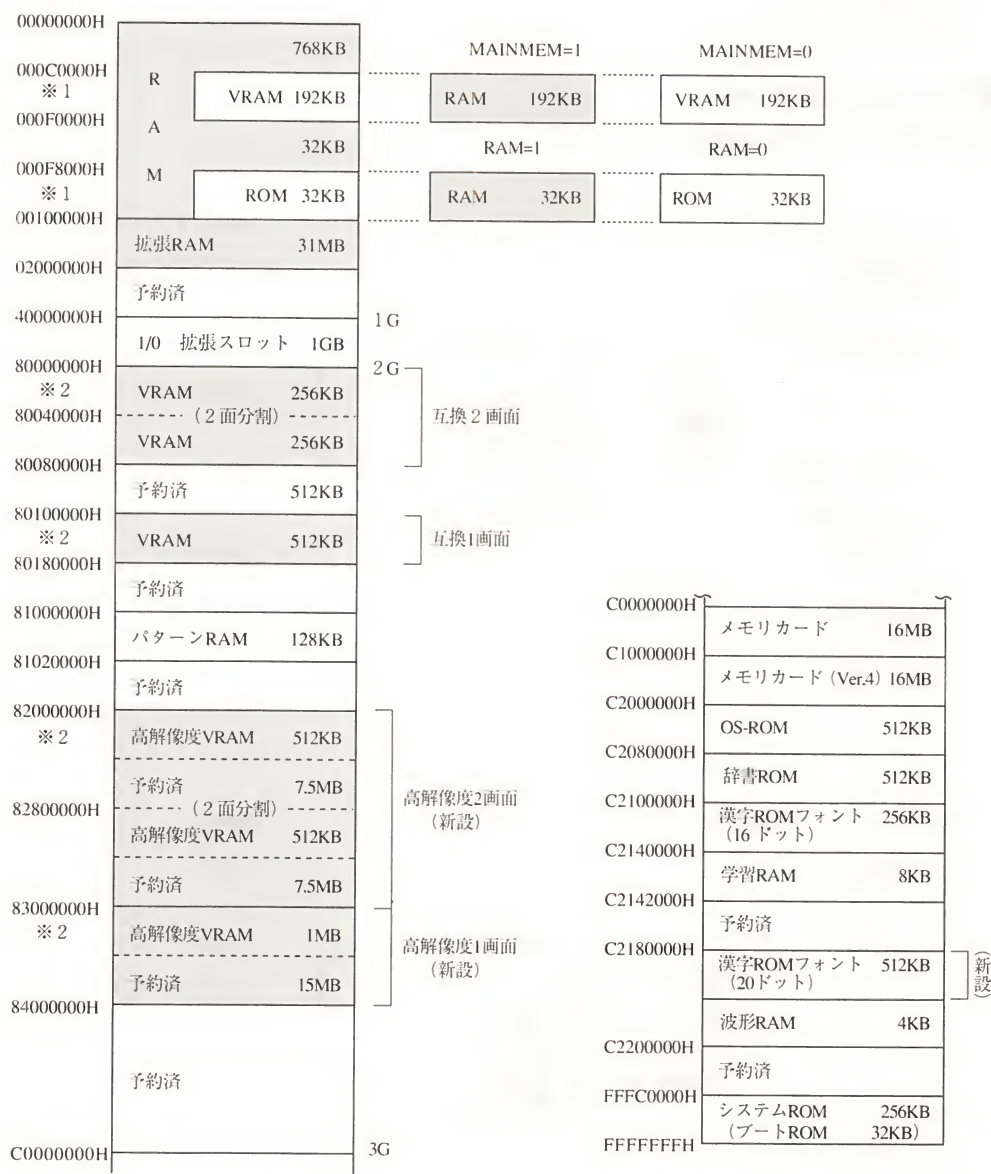
機種名	メモリ	FDD	ハードディスク
ME20	2MB	2	なし
ME170	2MB	2	170MB
MA20	4MB	2	なし
MA170	4MB	2	170MB
MA340	4MB	2	340MB
MX20	4MB	2	なし
MX170	4MB	2	170MB
MX340	4MB	2	340MB
MF20	4MB	2	なし
MF170W	6MB	2	170MB
Fresh	6MB	2	170MB
MA170W	8MB	2	170MB
MA340W	8MB	2	340MB
MX170W	8MB	2	170MB
MX340W	8MB	2	340MB

● MA，MX の外観

外観は HG，HR と基本的に同じです．各種コネクタの位置が異なっています．

● ME，MF，Freshの外観

次の図のように外観が変更されています．

●メモリマップ

FMTOWNS II ME, MF, Fresh を除き，高解像度画面サポートのため，VRAM の拡張(1MB)，高解像度アクセス領域の新設，20 ドット漢字フォント ROM 領域の追加によりメモリマップの変更があります.

図中の網かけ部分はキャッシュ使用可能領域です.

※1 000C0000～000EFFFF，000F8000～000FFFFFの領域は，裏RAMのみキャッシュ領域とする．
000C0000～000EFFFFの領域は，RAM→VRAMの切り替え時，000F8000～000FFFFFの領域はRAM→ROMの切り替え時にキャッシュをパージする．

※2 VRAM領域は，VCMENビット（I/O 05EEHのbit0）が1のときにのみキャッシュ領域となる．ただし，スプライト転送デジタイズおよびVIW時はキャッシュオフとなる．

●追加，および拡張されたI/Oとレジスタ

1. CPU識別レジスタ

CPU識別レジスタは，フォーマットの変更はありません．FMTOWNS II ME，MF，Fresh，MA，MXが追加されたことにより，図のビット構成(ME：0D02H，MF，Fresh：0F02H，MA：0B02H，MX：0C02H)が参照されます．この値は，シリアルROM識別情報のビット56〜71の内容と同じです．

なお，このレジスタのID15は，リセット直後は1となり，300μs後に0になります．0になったことを確認した上で参照してください．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0030H</td><td rowspan="4">CPU識別レジスタ</td><td rowspan="2">R</td><td colspan="5">MACHINE-ID</td><td colspan="3">CPU-ID</td></tr>
<tr><td>ID7</td><td>ID6</td><td>ID5</td><td>ID4</td><td>ID3</td><td>ID2</td><td>ID1</td><td>ID0</td></tr>
<tr><td rowspan="2">0031H</td><td rowspan="2">R</td><td colspan="8">MACHINE-ID</td></tr>
<tr><td>ID15</td><td>ID14</td><td>ID13</td><td>ID12</td><td>ID11</td><td>ID10</td><td>ID9</td><td>ID8</td></tr>
</table>

MACHINE-ID (bit15-3)：装置の種別を示す．下記のビット構成により識別を行う．

装置	ID15	ID14	ID13	ID12	ID11	ID10	ID9	ID8	ID7	ID6	ID5	ID4	ID3
FMR-60/50	不定								1	1	1	1	1
FMR-50S	不定								1	1	1	0	1
FMR-70	不定								1	1	1	1	0
FMTOWNS (1,2)	0	0	0	0	0	0	0	1	0	0	0	0	0
FMTOWNS (1F,2F,1H,2H)	0	0	0	0	0	0	1	0	0	0	0	0	0
FMTOWNS (10F,20F,40H,80H)	0	0	0	0	0	1	0	0	0	0	0	0	0
FMTOWNS (UX10,UX20,UX40)	0	0	0	0	0	0	1	1	0	0	0	0	0
FMTOWNS II (CX10,CX20 CX40,CX100)	0	0	0	0	0	1	0	1	0	0	0	0	0
FMTOWNS II (UG10,UG20 UG40,UG80)	0	0	0	0	0	1	1	0	0	0	0	0	0
FMTOWNS II (HG20,HG40 HG100)	0	0	0	0	1	0	0	0	0	0	0	0	0
FMTOWNS II (HR20,HR100 HR200)	0	0	0	0	0	1	1	1	0	0	0	0	0
FMTOWNS II (UR20,UR40 UR80)	0	0	0	0	1	0	0	1	0	0	0	0	0
FMTOWNS II (MA20,MA170 MA340)	0	0	0	0	1	0	1	1	0	0	0	0	0
FMTOWNS II (MX20,MX170 MX340)	0	0	0	0	1	1	0	0	0	0	0	0	0
FMTOWNS II (ME20,ME170)	0	0	0	0	1	1	0	1	0	0	0	0	0
FMTOWNS II (MF20,MF170 Fresh)	0	0	0	0	1	1	1	1	0	0	0	0	0

FMTOWNS 各機種の MACHINE-ID は，ID15-8 を使用し，ID7-3 が 0 のとき有効である．

CPU-ID(bit2-0)　：使用 CPU の種別を示す．下記のビット構成により識別を行う．

ID2	ID1	ID0	CPU
0	0	0	80286
0	0	1	80386
0	1	0	80486SX/DX
0	1	1	80386SX
1	0	0	予約済
1	0	1	予約済
1	1	0	予約済
1	1	1	予約済

2. CPU_MISC3レジスタ

DMACのアドレスレジスタA23→A24の桁上がりがあるかどうかを参照するビット(DMACMD)が追加されました．ME，MF，Fresh，MA，MXでは，0(桁上がりあり)となります．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0024H	CPU-MISC3レジスタ	R	ENPOFF (0)	RCREN (0)	CRT OFFEN	FTN (0)	POFFEN (0)	DMACMD (0)	1	1

ENPOFF(bit7)　：POFFEN(bit3)ビットが有効かどうかを示す．
　0 = POFFENビット有効
　1 = POFFENビット無効

RCREN(bit6)　：I/O FDA4H(リードコンパチレジスタ)が有効かどうかを示す．
　0 = リードコンパチレジスタが有効
　1 = リードコンパチレジスタが無効

CRTOFFEN(bit5)　：I/O 0022HのCRTPOWOFFビットが有効かどうかを示す．このビットは，電源断可能なCRTが接続されていることを示している．
　0 = CRTのソフト電源断が可能
　1 = CRTのソフト電源断が不可能

FTM(bit4)　：フリーランタイマ(I/0 0026, 0027H)の有無を示す．
　0 = フリーランタイマ有
　1 = フリーランタイマ無

POFFEN(bit3)　：I/O 0020HのPOFFビットが有効かどうかを示す．
　0 = ソフト電源断が可能
　1 = ソフト電源断が不可能

DMACMD(bit2)　：DMACのアドレスレジスタでA23からA24の桁上がりがあるかどうかを示す．
　0 = A23からA24の桁上がりする
　1 = A23からA24の桁上がりしない

3. メモリ容量レジスタ

このレジスタはMB単位で実装容量を示しており，従来は5ビット(1〜31MBまで表現可能)が使われていましたが，7ビットに拡張され，1〜127MBの表現ができるようになりました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05E8H	メモリ容量レジスタ	R	不定	SIZE6	SIZE5	SIZE4	SIZE3	SIZE2	SIZE1	SIZE0

SIZE6-0(bit6-0)　：メモリの実装容量を示す。ビットコードで1〜127MBまで表現される。

4. 最高速クロックレジスタ

装置の最高動作周波数を表すレジスタで，フォーマットの変更はありませんが，ME，MF，Fresh，MA，MX により，FCLKn 値が変更されました．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
05EDH	最高速クロックレジスタ	R	FCLKEN	FCLK6	FCLK5	FCLK4	FCLK3	FCLK2	FCLK1	FCLK0

FCLKEN(bit7)　：FCLK6-0 ビットが有効かどうかを示す．
0 ＝ FCLK6-0 ビットが有効
1 ＝ FCLK6-0 ビットが無効

FCLK6-0(bit6-0)　：装置の最高動作周波数を示す [MHz].
0～127MHz
ME:19H(25MHz)
MF:21H(33MHz)
MA:21H(33MHz)
MX:42H(66MHz)

5. VRAM キャッシュ制御レジスタ

VRAM キャッシュの動作を制御するレジスタについては，キャッシュのページとともに，CPU 外部のメモリなどとのデータの整合性を保つため再設定を行う I/O ビットが追加されました．

全ページ条件	I/O アドレス	ビット名	ビット動作
キャッシュ動作禁止時	00C0H 05ECH 05EEH	CMEN HSPEED VCMEN	1⇒0 1⇒0 1⇒0
000C0000H～000EFFFFH の領域を VRAM に切り替えたとき	0404H	MAINMEM	1⇒0
000F8000H～000FFFFFH の領域を ROM に切り替えたとき	0480H	RAM	1⇒0
スプライト転送を開始したとき (VRAM キャッシュ有効時のみ)	0452H (01H)	SPEN	0⇒1
ディジタイズを開始したとき (VRAM キャッシュ有効時のみ)	0442H (1CH)	ESYN ESM 0/1	1 0⇒1
VIW を開始したとき (VRAM キャッシュ有効時のみ)	047CH (0000H)	VIW ENBL	0⇒1
16M 色パック変換モードを切り替えたとき (VRAM キャッシュ有効時のみ)	047CH (0000H)	16M COLOR ENBL	0⇒1
他の VRAM をアクセスしたとき (VRAM キャッシュ有効時のみ) •VRAM1 (2 画面) ⇒ PLANE •VRAM2 (1 画面) ⇒ PLANE •高解像度 VRAM1 (2 画面) ⇒ PLANE •高解像度 VRAM2 (1 画面) ⇒ PLANE •VRAM1 (2 画面) ⟺ VRAM2 (1 画面) •VRAM1 (2 画面) ⟺ 高解像度 VRAM1 (2 画面) •VRAM1 (2 画面) ⟺ 高解像度 VRAM2 (1 画面) •VRAM2 (1 画面) ⟺ 高解像度 VRAM1 (2 画面) •VRAM1 (2 画面) ⟺ 高解像度 VRAM2 (1 画面) •高解像度 VRAM1 (2 画面) ⇔ 高解像度 VRAM2 (1 画面)	—	—	—

キャッシュ対象領域についてはメモリマップを参照のこと．
注）VIW, 高解像度関係については ME，MF，Fresh は除く．

6. アドレスレジスタ

従来機ではA23→A24の桁上がりがないのに対し，ME, MF, MA, MXでは桁上がりが行われます．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
00A4H	アドレスレジスタ	R/W	ADDRESS REG. (下位)							
			A7	A6	A5	A4	A3	A2	A1	A0
00A5H		R/W	ADDRESS REG. (中位)							
			A15	A14	A13	A12	A11	A10	A9	A8
00A6H		R/W	ADDRESS REG. (上位)							
			A23	A22	A21	A20	A19	A18	A17	A16
00A7H		R/W	ADDRESS REG. (最上位)							
			A31	A30	A29	A28	A27	A26	A25	A24

A31-0 ：DMA転送の開始アドレス(4GB空間)を指定する．従来機種では，A23からA24への桁上がりは行われないが，MA/MX/ME/MF/Freshでは，桁上がりが行われる．

7. FIFOモードレジスタ

新しく追加されたレジスタです．FUSARTが標準モードとFIFOモードを持っていることから，いずれかを選択するためのものです．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A0CH	FIFOモードレジスタ	R/W	FIFO MODE	0	0	0	0	0	0	0
				0	0	0	0	0	0	0

FIFOMODE(bit7) ：FUSARTに対して標準モードかFIFOモードかの指定を行う．
0 = 標準モードであることを示す(リセット時)
1 = FIFOモードであることを示す

8. FIFOステータスレジスタ

新設レジスタです．機種によりFIFOモードが使えるか(FUSARTが搭載されているか)否かを，FIFOINSビットで表します．ME, MF, MA, MXでは0(FIFO可能)が読み出されます．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0A0DH	FIFOステータスレジスタ	R/W	FIFOINS (0)	1	1	1	1	1	1	1

FIFOINS(bit7) ：USARTにFIFOモードが実装されていることを示す．
0 = FIFOモード機能が実装されていることを示す
1 = FIFOモード機能が実装されていないことを示す
MA/MX/ME/MF/Freshでは0(固定)．

9. FIFO 制御レジスタ

FUSART で FIFO モードを利用するときの，ステータス参照およびクリアするための新設レジスタです．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0A0EH</td><td rowspan="2">FIFO制御レジスタ</td><td>R</td><td>TCPL</td><td>TRSF FULL</td><td>TRSF EMPTY</td><td rowspan="2">TRSFC</td><td>RCVF FULL</td><td>RCVF EMPTY</td><td rowspan="2">RCVFC</td><td>不定</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

TCPL(bit7) ：全てのデータの送信が終了したことを示す最終データの最終ビット (調歩ではストップビット) を 1 ビット時間送信してからこのビットが 1 にセットされる．
0 ＝ 送信中
1 ＝ 送信完了

TRSFFULL(bit6) ：送信データ FIFO が一杯であることを示す．
0 ＝ 送信データ FIFO に空きがあることを示す
1 ＝ 送信データ FIFO が一杯であることを示す

TRSFEMPTY(bit5) ：送信データ FIFO が空であることを示す．
0 ＝ 送信データ FIFO にデータがあることを示す
1 ＝ 送信データ FIFO が空であることを示す

TRSFC(bit4) ：FUSART 内の送信データ FIFO をクリアする．
0 ＝ クリア解除 (リセット時)
1 ＝ 送信データ FIFO をクリアする
クリア時，1 のセット後 0 に戻すまでに 1μs 以上時間をあけること．

RCVFFULL(bit3) ：受信データ FIFO が一杯であることを示す．
0 ＝ 受信データ FIFO に空きがあることを示す
1 ＝ 受信データ FIFO が一杯であることを示す

RCVFEMPTY(bit2) ：受信データ FIFO が空であることを示す．
0 ＝ 受信データ FIFO にデータがあることを示す
1 ＝ 受信データ FIFO が空であることを示す

RCVFC(bit1) ：FUSART 内の受信データ FIFO をクリアする．
0 ＝ クリア解除 (リセット時)
1 ＝ 受信データ FIFO をクリアする
クリア時，1 のセット後 0 に戻すまでに 1μs 以上時間をあけること．

10. 高解像度機能レジスタ

高解像度機能が標準搭載されているかどうかを表すレジスタで，新設されました．HIRES ビットは，MA，MX では 0 (標準搭載) になっていますが，ME，MF，Fresh および従来機では 1 (搭載されていない) である点に注意が必要です．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0470H	高解像度機能レジスタ	R	HIRES	1	1	1	1	1	1	1

HIRES(bit7)　：本装置に高解像度機能が標準搭載されているか否かを示す.
0 = 高解像度機能が標準搭載されている
1 = 高解像度機能が標準搭載されていない
MA/MX では 0(固定).

11. VRAM 容量レジスタ

新設レジスタですが, 高解像度機能レジスタで HIRES ビットが 0 (高解像度機能標準搭載) のときのみ有効です. VSIZEn は, 1~16 で MB 単位に VRAM の容量を表します. MA, MX では 1(固定) で, 1MB 実装されていることを示します.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0471H	VRAM容量レジスタ	R	0	0	0	VSIZE4 (0)	VSIZE3 (0)	VSIZE2 (0)	VSIZE1 (0)	VSIZE0 (1)

VSIZE4-0(bit4-0)　：VRAM の容量を示す. (1~16MB)
MA/MX では, 1MB 固定.
このレジスタは, HIRES ビットが 0 のとき有効.

12. 画像出力制御アドレスレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0472H	画像出力制御アドレスレジスタ	R/W	RA7	RA6	RA5	RA4	RA3	RA2	RA1	RA0
0473H		R/W	RA15	RA14	RA13	RA12	RA11	RA10	RA9	RA8

RA15-0(bit15-0)　：画像出力制御用レジスタでアクセスする間接レジスタアドレスを指定する.

13. 画像出力制御データレジスタ

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0474H	画像出力制御データレジスタ	R/W	RD7	RD6	RD5	RD4	RD3	RD2	RD1	RD0
0475H		R/W	RD15	RD14	RD13	RD12	RD11	RD10	RD9	RD8
0476H		R/W	RD23	RD22	RD21	RD20	RD19	RD18	RD17	RD16
0477H		R/W	RD31	RD30	RD29	RD28	RD27	RD26	RD25	RD24

RD31-0(bit31-0)　：画像出力制御用データレジスタ.

14. 新 PCM 音源 AD/DA バンク切替レジスタ

新 PCM 音源のための新設レジスタのひとつです．このレジスタは，新 PCM 音源が DMA 機能を使うとき，アドレスとしてベース側かカレント側かを選択する機能のほかに，新 PCM 音源の存在確認 (AD/DA 機能確認) と，割り込み要因解析のためのフラグを参照するのに用います．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="3">0510H</td><td rowspan="3">バンク切替レジスタ</td><td rowspan="2">R</td><td rowspan="3">ADDA機能(0)</td><td colspan="3">割込要因判別フラグ</td><td colspan="3" rowspan="2">不定</td><td rowspan="3">ベース/カレント切り替え</td></tr>
<tr><td>音声入力</td><td>バッファ</td><td>DMA</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

AD-DA 機能確認情報 (bit7)：AD-DA 機能の有無が，当ビットのアクセスで確認できる．
　0 ＝ AD-DA 機能有り
　1 ＝ AD-DA 機能なし
　MA/MX/ME/MF は 0(固定)

割り込み要因判別フラグ (bit6-4)：AD-DA 機能から発生した割り込みがどの要因によるものかを示すフラグで，マスクされている要因については常に 0 となり，実際に発生した割り込み要求についてのみ 1 となる．
　0 ＝ 割り込み要求なし
　1 ＝ 割り込み要求有り

ベース/カレント切替 (bit0)：DMA カウンタ, DMA アドレスの各レジスタをアクセスする際，ベース側かカレント側かを切り替える．
　0 ＝ READ … カレント選択 (リセット時)
　　　WRITE … ベース/カレント共
　1 ＝ READ/WRITE ともにベースのみ選択

15. 新 PCM 音源 DMA ステータスレジスタ

新 PCM 音源のための新設レジスタのひとつです．DMA の機能設定と，転送終了フラグから成っています．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">0511H</td><td rowspan="2">DMAステータスレジスタ</td><td>R</td><td colspan="3">不定</td><td rowspan="2">AUTO INIT</td><td colspan="2">不定</td><td rowspan="2">転送終了割込許可</td><td rowspan="2">転送終了フラグ</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

AUTOINIT(bit4)：DMA 転送でオートイニシャライズを行うか否かを設定する．
オートイニシャライズを行うと，転送終了時に DREQ マスクはセットされず，アドレスカウンタの各カウントにはベース設定値が読み込まれる．
　0 ＝ オートイニシャライズ禁止 (リセット時)
　1 ＝ オートイニシャライズを行う

転送終了割り込み許可 (bit1)：転送終了フラグが 1 のとき，割り込みを発生させるか否かを設定する．
　0 ＝ 割り込みを発生させない (リセット時)
　1 ＝ 割り込みを発生する

転送終了フラグ (bit0)：DMA 転送が終了すると 1 になる．
この bit をリセットするときは 1 を WRITE する．
　0 ＝ 通常時
　1 ＝ DMA 転送終了時

16. 新PCM音源DMAカウンタレジスタ

新PCM音源のための新設レジスタのひとつで，DMA転送を行うときの転送ワード数を設定するレジスタです．DMA動作はワード単位に行われるので，8ビットモノラルの奇数バイト再生時には，データの最後に80Hを加えて偶数バイトにしなければなりません．同じレジスタで，ベース/カレントの双方に用いられ，その切り替えはAD/DAバンク切替レジスタのビット0で行います．

DMA転送中，このレジスタを読み出してもカレントのカウンタ値は保証されないので注意が必要です．また，DMA動作はワード単位に行われるので，8ビットモノラルの奇数バイト再生時には，データの最後に80Hを加えて偶数バイトにしなければなりません．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0512H	DMAカウンタレジスタ	R/W	COUNT REG. (下位)							
			C7	C6	C5	C4	C3	C2	C1	C0
0513H		R/W	COUNT REG. (上位)							
			C15	C14	C13	C12	C11	C10	C9	C8

DMA転送を行う際の転送ワード数を設定する．なお，DMA転送はREC/PLAYいずれの場合にもワード転送で行われ，8ビットデータの場合は以下のように扱われる．

モノラル8ビット:C15～C8　次のデータ
　　　　　　　:C7～C0　後のデータ
ステレオ8ビット:C15～C8　Rchのデータ
　　　　　　　:C7～C0　Lchのデータ

このレジスタにはベース/カレントがあり，バンク切替レジスタの内容によってアクセスされるレジスタが決定される．
ベースレジスタは，設定された値を新たな設定が行われるまで保持し，オートイニシャライズ時にはその値をカレントレジスタへ転送する．
カレントレジスタは，1ワード転送するたびに1だけカウントダウンされる．
このレジスタには事前に転送回数−1の値を設定する．この値は，DMA転送終了時にはFFFFHとなる．また，FFFFHを設定したときは65536ワード転送される．

17. 新PCM音源DMAアドレスレジスタ

新PCM音源のための新設レジスタのひとつです．DMA転送を行う際の転送開始アドレスを設定するのに用いられます．同じレジスタで，ベース／カレントの双方に用いられ，その切り替えはAD/DAバンク切替レジスタのビット0で行います．

DMA転送中，このレジスタを読み出してもカレントアドレスの値は保証されないので注意が必要です．また，DMA動作はワード単位に行われるので，奇数アドレスの設定はできません．A0の位置に1を書いても0とみなされます．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0514H	DMAアドレスレジスタ	R/W	ADDRESS REG. (下位)							
			A7	A6	A5	A4	A3	A2	A1	A0
0515H		R/W	ADDRESS REG. (中位)							
			A15	A14	A13	A12	A11	A10	A9	A8
0516H		R/W	ADDRESS REG. (上位)							
			A23	A22	A21	A20	A19	A18	A17	A16
0517H		R/W	ADDRESS REG. (最上位)							
			A31	A30	A29	A28	A27	A26	A25	A24

DMA 転送を行う際の転送開始アドレスを設定する.
このレジスタにはベース/カレントがあり，バンク切替レジスタの内容によってアクセスされるレジスタが決定される.
ベースレジスタは，設定された値を新たな設定が行われるまで保持し，オートイニシャライズ時にはその値をカレントレジスタへ転送する.
カレントレジスタは1ワード転送するたびに2ずつカウントアップされる.

18. 新 PCM 音源クロック設定レジスタ

新 PCM 音源のための新設レジスタのひとつで，サンプリングレートを参照／設定します．通常はリセットされたときの 19.2kHz のまま使いますが，CLKn 値を書き込むことによって任意の値に変更できます.

ただし，そのことにより，既存のサンプリングレート固定値 (19.2kHz) も影響を受けるので注意が必要です．そこで，新 PCM 音源の機能を使用した後は，新 PCM 音源システムコントロールレジスタの ADDA リセットを使い，使用前の状態に復元するようにします.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0518H	クロック設定レジスタ	R	不定				サンプリングレート設定			
							CLK3	CLK2	CLK1	CLK0
		W	0	0	0	0				

CLK3-0(bit3-0)　：サンプリングレート設定
サンプリングレートを以下のように設定する．設定禁止の値を指定した場合の動作は保証しない.

CLK3	CLK2	CLK1	CLK0	サンプリングレート
0	0	0	0	48.0000KHz
0	0	0	1	44.1000KHz
0	0	1	0	32.0000KHz
0	0	1	1	22.0500KHz
0	1	0	0	19.2000KHz(リセット時)
0	1	0	1	16.0000KHz
0	1	1	0	11.0250KHz
0	1	1	1	9.6000KHz
1	0	0	0	8.0000KHz
1	0	0	1	5.5125KHz
1	0	1	0	24.0000KHz
1	0	1	1	12.0000KHz
1	1	0	0	設定禁止
1	1	0	1	設定禁止
1	1	1	0	設定禁止
1	1	1	1	設定禁止

19. 新PCM音源モード設定レジスタ

新PCM音源のために新規に追加されたレジスタで，ステレオ／モノラルの選択やビットモードなどの設定を行うためのものです．この中で，変換レベルは，既存のサンプリングLSB位置にも影響するので，新PCM音源の機能を使用した後は，新PCM音源システムコントロールレジスタのADDAリセットを使い，使用前の状態に復元するようにします．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
0519H	モード設定レジスタ	R	不定		SND/WAV切り替え	ST/MONO切り替え	16/8切り替え	変換レベル設定 bit2	変換レベル設定 bit1	変換レベル設定 bit0
		W	0	0						

このレジスタはバイト/ワードアクセス可能で，前述のクロック設定レジスタと一括してREAD/WRITEすることができる．

SND/WAV切り替え：データフォーマットを設定する．
(bit5)
0 = WAV形式 ←リセット時
1 = SND形式

入出力電圧	SND	WAV8	WAV16
+2Vrms ≀ +0Vrms	FE ≀ 80	FF ≀ 80	7FFF ≀ 0000
−0Vrms ≀ −2Vrms	01 ≀ 7F	7F ≀ 00	FFFF ≀ 8000

注意：SND形式を選択するとbit4，3は無効となり8bitモノラルが選択されたものと認識される．

ST/MONO切り替え：データフォーマットを設定する．
(bit4)
0 = モノラル ←リセット時/SND形式設定時
※モノラルデータはLchとRchの和となる．
1 = ステレオ

16/8 ビット切り替え：データフォーマットを設定する.
(bit3)
　　0 ＝ 8 ビット　←リセット時/SND 形式設定時
　　1 ＝ 16 ビット

変換レベル設定：16bit データから 8 ビットに変換するとき，16 ビットデータのどこを LSB
(bit2〜0)　にするかを次のように設定する.

bit2	bit1	bit0	LSB 設定
0	0	0	bit1
0	0	1	bit2
0	1	0	bit3
0	1	1	bit4
1	0	0	bit5
1	0	1	bit6
1	1	0	bit7
1	1	1	bit8

この設定は録音時のみ有効で，8bit データの再生時は下位 8bit が 0 の 16bit データとして再生される.
従来の音源との互換性維持のため，ブートを行うと bit2〜bit0 が〝101〞に設定されるが，システムコントロールレジスタ (051AH) の bit7 でリセットを行うと〝111〞になってしまうので，再設定が必要.

設定値の例：19.2kHz サンプリング 16bit データの bit13〜7 を SND 形式で録音する.
　クロック設定:00000100b＝04h
　モード設定　:00100110b＝26h

20.　新 PCM 音源システムコントロールレジスタ

新 PCM 音源のために新規に追加されたレジスタです．主に，新 PCM 音源の動作を指示します.

新 PCM 音源クロック設定レジスタでクロックの設定を行った後や，新 PCM 音源モード設定レジスタでサンプリング LSB の設定を行った後，既存の PCM に切り替えるときは，ADDA リセットを使い，使用前の状態に復元するようにします.

<table>
<tr><th>I/0アドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="3">051AH</td><td rowspan="3">システムコントロール
レジスタ</td><td rowspan="2">R</td><td rowspan="3">ADDA
リセット</td><td rowspan="2">不定</td><td rowspan="3">レベル
モニタ</td><td rowspan="3">REC/
PLAY</td><td rowspan="2">不定</td><td colspan="3">割り込みレベル設定</td></tr>
<tr><td rowspan="2">bit2</td><td rowspan="2">bit1</td><td rowspan="2">bit0</td></tr>
<tr><td>W</td><td>0</td><td>0</td></tr>
</table>

ADDA リセット：AD-DA 機能のリセットを行うときは，このビットに 1 を書き込む.
(bit7)　その後，約 2μs でリセットは解除され，自動的に 0 となる.
本リセットにより，AD-DA 制御の各レジスタはリセット時の値になる.
他のビットを設定したいときには，このビットは 0 を WRITE する.
　0 ＝ READ　　通常状態
　　　 WRITE　無効
　1 ＝ READ　　リセット中
　　　 WRITE　リセット実行

レベルモニタ (bit5)：データポートレジスタの機能設定を次のように行う.

bit5	データポート機能
0	レベルモニタ
1	ソフト転送ポート

各機能についての詳細は，データポートレジスタを参照のこと.

REC/PLAY(bit4)　：このビットはREC/PLAYの切り替えを行うスイッチで，DMAC/バッファの転送方向を決定する．
　0 = PLAY　←リセット時
　1 = REC

割り込みレベル設定(bit2〜0)：割り込みレベルの設定を次のように行う．

bit2	bit1	bit0	割り込み設定
0	0	0	INT 4
0	0	1	INT 5
0	1	0	INT 10
0	1	1	INT 14
1	0	0	INT 15
1	0	1	割り込み禁止
1	1	0	割り込み禁止
1	1	1	割り込み禁止 ← リセット時

21. 新PCM音源バッファコントロールレジスタ

新PCM音源のために新規に追加されたレジスタです．新PCM音源のエラーフラグおよび解除兼用ビットや，バッファの割り込み許可ビットなどを持ちます．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
051BH	バッファコントロールレジスタ	R	バッファ割り込み許可	不定	バッファエラー		バッファステータス/INIT			
					OVER	UNDR	bit3	bit2	bit1	bit0
		W		0						

バッファ割り込み許可(bit7)：バッファエラー時に割り込みを発生させるか否かを設定する．
　0 = 割り込みを発生させない ← リセット時
　1 = 割り込みを発生する

バッファエラー(OVER)(bit5)：バッファのオーバーラン発生を示すフラグで，クリアするときは1をWRITEする．
オーバーラン発生中は，バッファに入りきれないデータはそのまま捨てられる．
　0 = 通常状態 ← リセット時
　1 = READ　オーバーラン発生
　　　WRITE クリア

バッファエラー(UNDR)(bit4)：バッファのアンダーラン発生を示すフラグで，クリアするときは1をWRITEする．
アンダーラン発生中は，アンダーラン発生直前のデータが保持され，出力される．
　0 = 通常状態 ← リセット時
　1 = READ　アンダーラン発生
　　　WRITE クリア

バッファステータス/INIT(bit3〜0)：
READ時
　現在データが入っているバッファの段数を0000b〜1111bで表す．
WRITE時
　1111bをWRITEすることでバッファをイニシャライズする．
　その他のデータは無効となる．

22. 新PCM音源録音／再生制御レジスタ

新PCM音源のために新規に追加されたレジスタで，録音／再生にかかわる参照フラグや設定ビットを持っています．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">051CH</td><td rowspan="2">録音/再生制御レジスタ</td><td>R</td><td>A/D RDY</td><td>D/A RDY</td><td rowspan="2">音声入力割り込み許可</td><td rowspan="2">音声入力フラグ</td><td colspan="2">不定</td><td rowspan="2">DREQマスク</td><td rowspan="2">バッファマスク</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

A/D RDY フラグ (bit7)：A/Dコンバータが使用可能状態か否かを示す．
リセット後とサンプリングレート設定後には，A/Dコンバータの動作が不安定となる場合があり，その期間はこのフラグが1となる．よって，このフラグが1となっている期間は録音を行ってはならない．
5秒以上待ってもこのフラグが0とならないときはA/Dコンバータの不良と考えられる．
0 = 録音可能
1 = 録音禁止

D/A RDY フラグ (bit6)：D/Aコンバータが使用可能状態か否かを示す．
リセット後とサンプリングレート設定後には，D/Aコンバータの動作が不安定となる場合があり，その期間はこのフラグが1となる．よって，このフラグが1となっている期間は再生を行ってはならない．
5秒以上待ってもこのフラグが0とならないときはD/Aコンバータの不良と考えられる．
0 = 再生可能
1 = 再生禁止

音声入力割り込み許可 (bit5)：音声入力フラグが1の時に割り込みを発生させるか否かを設定する．
0 = 割り込みを発生させない．←リセット時
0 = 割り込みを発生させる．

音声入力フラグ (bit4)：録音レベルが録音ピークモニタに設定した閾値と等しいか，それを超えたことを示す．
このビットは1をWRITEしたときにクリアされる．
0 = READ　録音レベルが録音ピークモニタに設定した閾値より小さい
　　WRITE　無効
1 = READ　録音レベルが録音ピークモニタに設定した閾値と等しいか，それを超えたことを示す
　　WRITE　クリア

DREQマスク (bit1)：このビットは，バッファからDMAリクエストが来たときDMA転送を行うか否かを設定する．
このビットが1のとき，バッファからDMAリクエストがきてもDMA転送は行われない．
オートイニシャライズを行わないDMA転送の終了時には，このビットは自動的に1がセットされる．
0 = DMA転送を行う
1 = DMA転送を行わない(リセット時)

バッファマスク (bit0)：このビットは転送バッファの動作を制御する．
bit1，bit0が0だったらDMA転送が行われる．
bit1が1でbit0が0の場合，データポートを使用したソフト転送モードとなる．
0 = バッファ動作許可
1 = バッファ動作禁止(リセット時)

23. 新PCM音源ピークモニタレジスタ

新PCM音源のために新規に追加されたレジスタです．入力信号のピーク値をとらえ，以前の値より大きいと更新していきます．この動作はリセット直後から開始しており，そのままでは以前からの値を保持しているので，モニタを開始したいときには，次に述べるトリガレベルレジスタのコントロールビットを使って，クリアした上で使用する必要があります．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
051DH	録音ピークモニタレジスタ	R	LVL OVER	L6	L5	L4	L3	L2	L1	L0

このレジスタをREADすることで，現在保持されているピーク値を読み取ることができる．

LVL OVERフラグ (bit7)：A/Dコンバータへの入力信号が過大であることを示す．
0 = READ　通常状態
1 = READ　入力過大

ピークモニタフラグ (bit6〜0)：このレジスタにて扱われるデータは，モード設定レジスタで指定されたデータ形式を元に，上位8ビットから絶対値を求めて，次のように表す．

入力レベル	データ
+PEAK	7FH
≀	≀
0V	00H
≀	≀
−PEAK	7FH

入力データが現在保持されているピーク値より大きいときには，その入力データを保持する．
このレジスタは録音制御レジスタの値によらず，リセット終了時から常に動作しているので，使用する前には必ずクリアすること．

24. 新PCM音源トリガレベルレジスタ

新PCM音源のために新規に追加されたレジスタです．入力レベルがこのレジスタで設定されたLn値を越えると割り込みが発生します．また，ピークモニタレジスタをリセットする機能も持っています．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
051DH	トリガレベルレジスタ	W	LVL OVER	L6	L5	L4	L3	L2	L1	L0

このレジスタに割り込みを発生させる閾値をセットすることができる.

コントロールビット：　0 ＝ WRITE　割り込みを発生させる閾値をセット
(bit7)　　1 ＝ WRITE　LVL OVER フラグおよびピークモニタフラグをクリアする

トリガレベル　：割り込みを発生させる閾値をセットする.
(bit6〜0)　このレジスタにて扱われるデータは，モード設定レジスタで指定されたデータ形式を元に，上位 8 ビットからの絶対値を指定すること.

入力レベル	データ
+PEAK	7FH
≀	≀
0V	00H
≀	≀
−PEAK	7FH

25. 新 PCM 音源データポート

新 PCM 音源のために新規に追加されたレジスタで，システムコントロールレジスタのビット 5(レベルモニタ) により，0 のときレベルモニタレジスタ (A/D コンバータ出力現在値を参照)，1 のときソフト転送ポートレジスタ (DMA を使わずソフトウェアで転送するときの転送ポート) として働きます.

レベルモニタレジスタは，左右同時に参照できますが，上位 8 ビットのみが対象になっていることに注意が必要です.

データポート (0)

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
051EH	レベルモニタレジスタ	R	LVL OVER <L>	L6	L5	L4	L3	L2	L1	L0
051FH		R	LVL OVER <R>	R6	R5	R4	R3	R2	R1	R0

レベルモニタ 0 のときレベルモニタのデータポートになる.
BYTE/WORD アクセス可能で，READ オンリーである.
WORD アクセスしたとき，16 ビットレジスタのフォーマットは LVLOVER<R>(MSB)〜L0(LSB) の順となる.
レベルモニタレジスタは，A/D コンバータから出力される音の現在のレベルを示す.
051EH 番地は LEFT チャネルの音のレベルを，051FH 番地は右チャネルの音のレベルを示す.

LVL OVER フラグ　：A/D コンバータへの入力信号が過大であることを示す.
(bit7)　　0 ＝ READ　通常状態
　　1 ＝ READ　入力過大

レベルモニタ (bit6〜0)　：レベルモニタレジスタにて扱われるデータは，モード設定レジスタで指定されたデータ形式を元に，上位8bitから絶対値を求めて次のように表す．

入力レベル	データ
＋PEAK	7FH
≀	≀
0V	00H
≀	≀
−PEAK	7FH

レベルモニタレジスタは録音制御レジスタの値によらず，リセット終了時から常に動作している．

ソフト転送ポートレジスタは，録音時はADコンバータ出力の読み出し，再生時はDAコンバータへの書き込みを行うためのものであり，それ以外の動作は無効です．

参考までに，AD/DAプログラム例として，(1)新PCM音源の初期化，(2)録音時のDMA転送シーケンス，(3)再生時のソフト転送シーケンスの各場面について，それぞれフローチャートで示します．

データポート(1)

I/0アドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
051FH	ソフト転送ポートレジスタ	R	D15	D14	D13	D12	D11	D10	D9	D8
051EH		W	D7	D6	D5	D4	D3	D2	D1	D0

レベルモニタ1のときソフトウェア転送用のデータポートになる．
051EHからのWORDアクセス専用で，再生時のREADは不定，録音時のWRITEは無効となる．
16ビットレジスタでのフォーマットは，D15(MSB)〜D0(LSB)の順となる．
データ形式はモード設定レジスタで指定された形式で，以下の方法でアクセスする．

モノラル8ビット　：D15〜D8　後のデータ
D7〜D0　後のデータ

ステレオ8ビット　：D15〜D8　Rchのデータ
D7〜D0　Lchのデータ

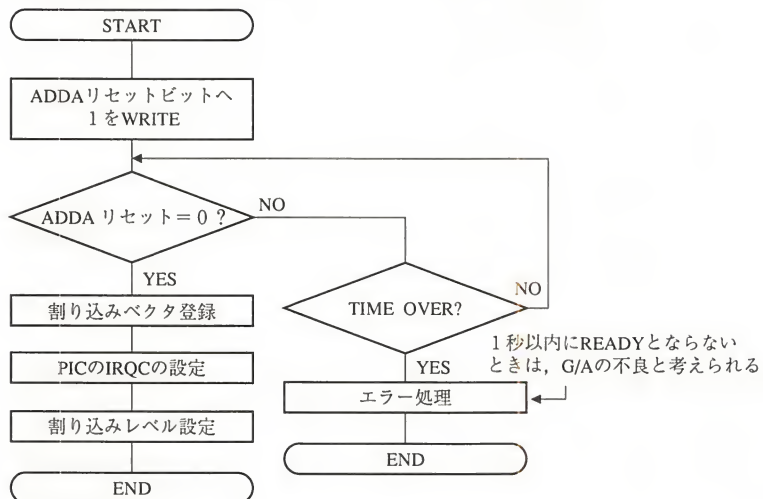
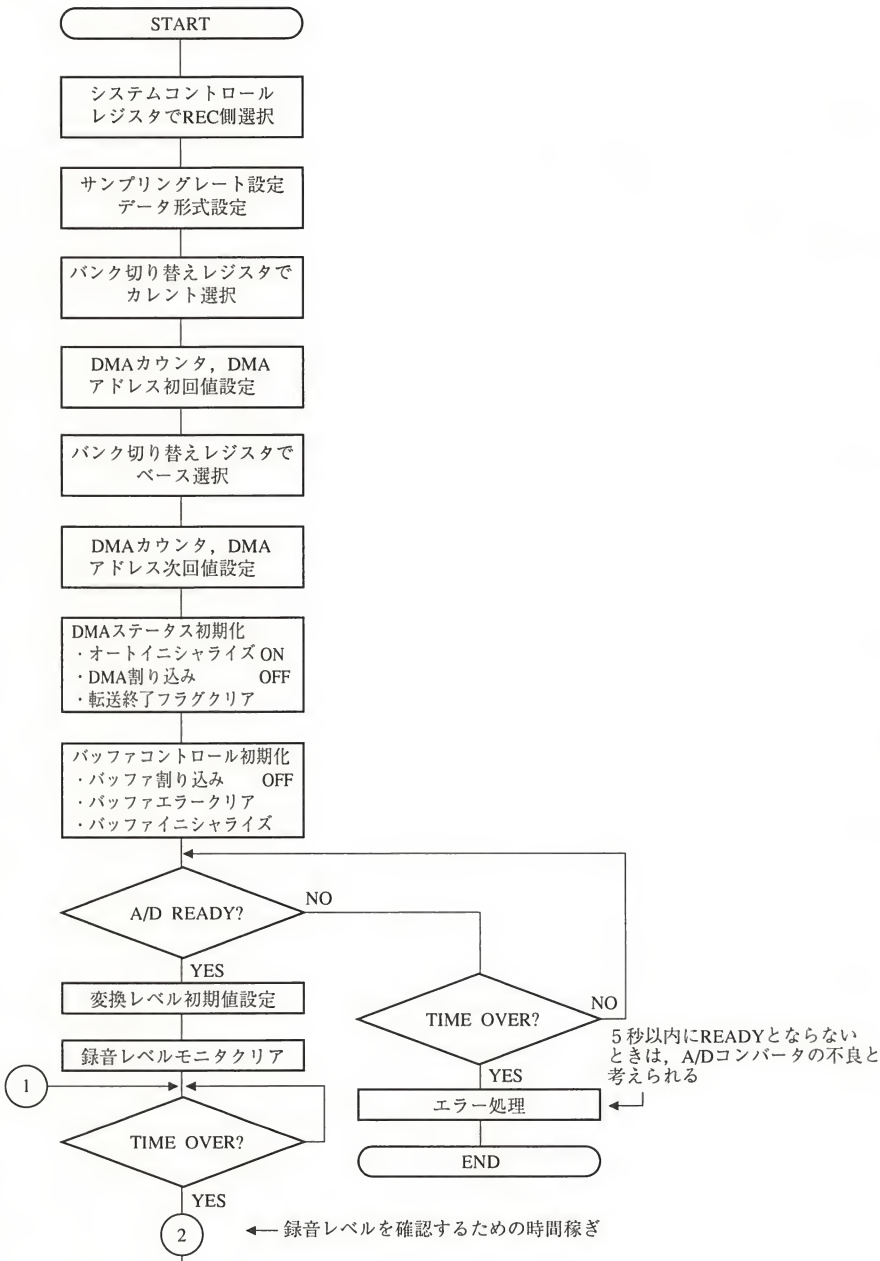
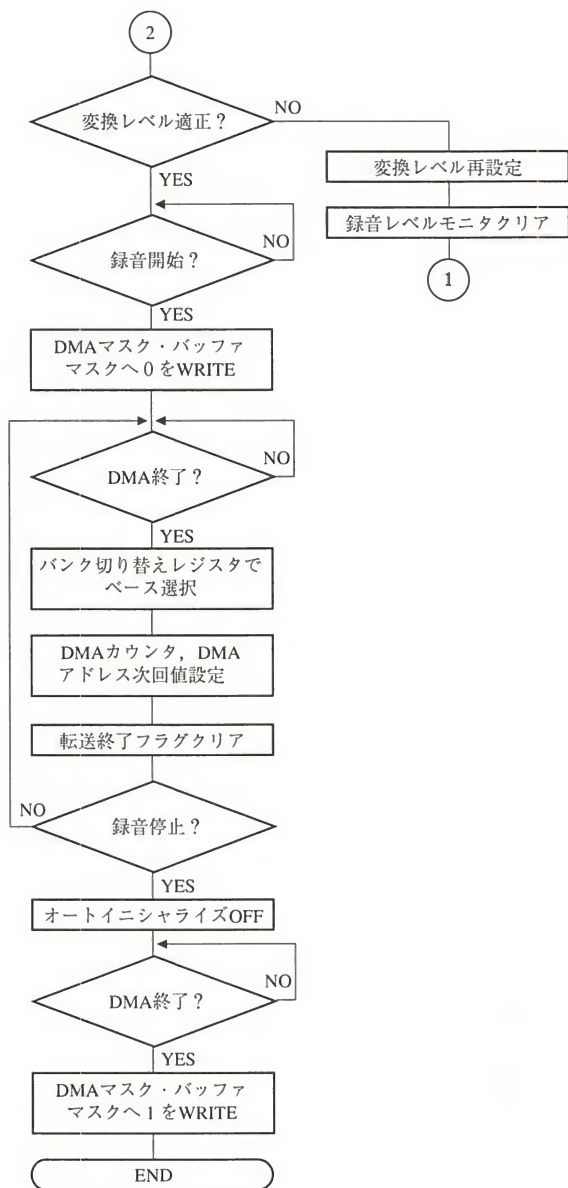
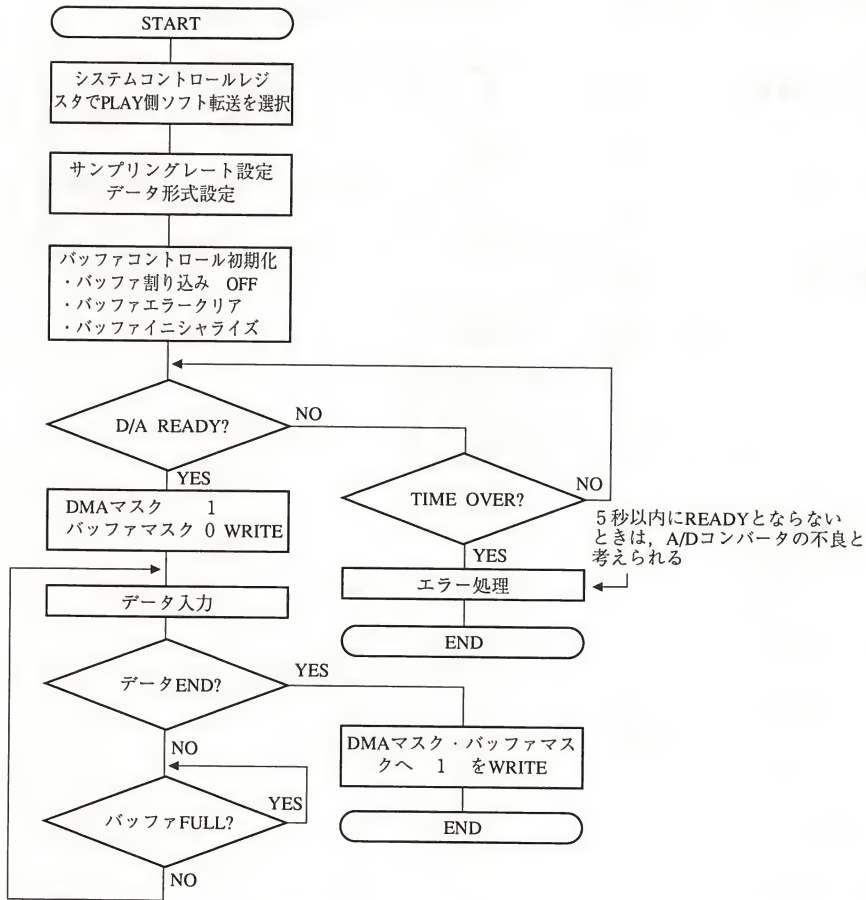
モノラル16ビット　：D15〜D0で1データ

ステレオ16ビット　：Lch・Rch……の順で，片チャネルずつ逐次

（1）　初期化シーケンス

START
ADDAリセットビットへ
1をWRITE
ADDA リセット＝0 ？
NO
YES
割り込みベクタ登録
PICのIRQCの設定
割り込みレベル設定
END
TIME OVER?
NO
YES
1秒以内にREADYとならない
ときは，G/Aの不良と考えられる
エラー処理
END

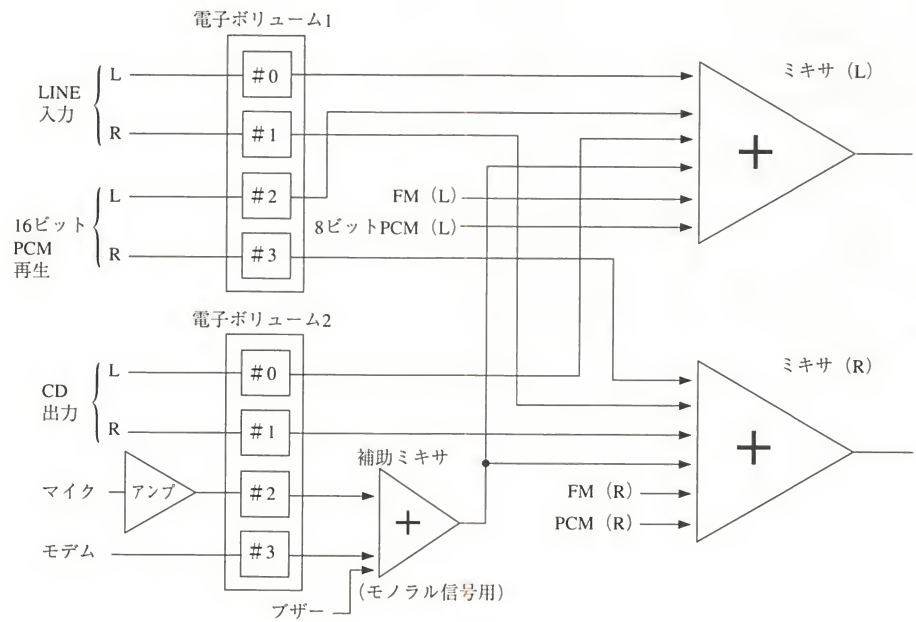
(2) DMA転送シーケンス

2
変換レベル適正？
NO
YES
変換レベル再設定
録音レベルモニタクリア
1
録音開始？
NO
YES
DMAマスク・バッファマスクへ0をWRITE
DMA終了？
NO
YES
バンク切り替えレジスタでベース選択
DMAカウンタ，DMAアドレス次回値設定
転送終了フラグクリア
録音停止？
NO
YES
オートイニシャライズOFF
DMA終了？
NO
YES
DMAマスク・バッファマスクへ1をWRITE
END

（3） ソフト転送シーケンス

26. ボリューム 1COM レジスタ

チャンネルの選択で，新 PCM 音源再生出力からの左右の信号が追加されました.

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04E1H	ボリューム1COMレジスタ	R	不定			C32	C0	EN	CH1	CH0
		W	0	0	0					

CH1-0(bit1-0)　：チャネルを選択する.

CH1	CH0	チャネル
0	0	ライン入力の左
0	1	ライン入力の右
1	0	AD-DA の左
0	0	AD-DA の右

27. CD-ROM機能レジスタ

CD-ROMドライブが倍速化されたことなどにより，新設されたレジスタです．MA，MX，ME，MF，Freshから対象となります．

I/Oアドレス	レジスタ名	R/W	7	6	5	4	3	2	1	0
04B0H	CD-ROM機能レジスタ	R	*ADP	*DS	*MS	*ME	1	1	1	1

*ADP(bit7)　：ADPCM再生機能が搭載されていることを示す．
　0 = ADPCM再生機能あり
　1 = ADPCM再生機能なし

*DS(bit6)　：倍速CD-ROMドライブが搭載されていることを示す．
　0 = 倍速CD-ROMドライブが搭載されている
　1 = 倍速CD-ROMドライブが搭載されていない
MA/MX/ME/MF/Freshでは，0(固定)

*MS(bit5)　：CD-ROMのマルチセッション対応がされていることを示す．
　0 = CD-ROMのマルチセッション対応がされている
　1 = CD-ROMのマルチセッション対応がされていない
MA/MX/ME/MF/Freshでは，0(固定)

*ME(bit4)　：CD-ROMのマウント/イジェクト対応がされていることを示す．
　0 = CD-ROMのマウント/イジェクト対応がされている
　1 = CD-ROMのマウント/イジェクト対応がされていない
MA/MX/ME/MF/Freshでは，0(固定)

28. CD-ROMキャッシュ制御レジスタ

HRから追加されたレジスタですが，MA，MX，ME，Freshからは，CACHEENビットが倍速動作の設定にも利用されるようになりました．

<table>
<tr><th>I/Oアドレス</th><th>レジスタ名</th><th>R/W</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td rowspan="2">04C8H</td><td rowspan="2">CD-ROMキャッシュ制御レジスタ</td><td>R</td><td>CACHE</td><td colspan="6">不定</td><td rowspan="2">CACHE EN</td></tr>
<tr><td>W</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>

CACHE(bit7)　：CD-ROMキャッシュ機能が搭載されていることを示す．
　0 = CD-ROMキャッシュあり
　1 = CD-ROMキャッシュなし

CACHEEN(bit0)　：CD-ROMキャッシュを有効にするかどうかを選択する(bit7=0のとき有効)．
このビットは，HSPEED (I/O 05ECHのbit0)と連動してセット/リセットされる．
また，MA，MX，ME，MF，Fresh以降の機種では，倍速動作の設定もされる．
　0 = CD-ROMキャッシュ/倍速動作を無効にする(リセット時)
　1 = CD-ROMキャッシュ/倍速動作を有効にする

索 引

ハードウェア

ハードウェア概要

80386CPU

80486CPU

CPU 近傍のハードウェア

割り込みコントローラ

DMA コントローラ

プログラマブルタイマ

リアルタイムクロック

CPU 近傍のレジスタ

表示システム

オーディオシステム

CD-ROM ドライブ〔CD ドライブ〕

キーボード

パッド，マウス

プリンタ

フロッピィディスクドライブ

ハードディスク

RS-232C インタフェース

BIOS

BIOS 全般

グラフィック BIOS

スプライト BIOS

マウス BIOS

フォントBIOS

サウンドBIOS

CD-ROM BIOS

キーボード BIOS

ディスク BIOS

プリンタ BIOS

時計をサポートする BIOS

RS-232C BIOS

ブザー BIOS

割り込み管理 BIOS

サービスルーチン，拡張サービスルーチン

システム情報 BIOS

音源割り込み管理 BIOS

MIDI マネージャBIOS

付 録

■著者略歴

千葉憲昭（ちばのりあき）

1946年　(世界最初のコンピュータ ENIAC 誕生年)北海道に生まれる

1969年　福岡工大電子工学科卒業

　　　　北海道総務部電子計算課勤務

1984年　著述家に転向、現在に至る(札幌在住)

　　　　(財)札幌エレクトロニクス・センター運営委員

著　書　「マイコンピュータNo.6，21」(CQ 出版社)

　　　　「6809マシン語スタディ」(CQ 出版社)

　　　　「BASIC ユーティリティ・プログラミング」(CQ 出版社)

　　　　「FM シリーズいざという時の事典」(ナツメ社)

　　　　「68000システムの製作全科（上、下)」(技術評論社)

　　　　「X68000 ベスト・プログラミング入門」(技術評論社)

　　　　外多数

■参考文献

1）杉原英文(インテルジャパン),「メモリ管理ユニットを内蔵した32ビット・マイクロプロセサ80386」,『日経エレクトロニクス』, 1985年11月4日号, No.381, PP.275-310

2）『80386プログラマーズ・リファレンスマニュアル』, インテルジャパン

3）『YM2203アプリケーションマニュアル』, ヤマハ

4）菅原清文（インテルジャパン),「次世代 32 ビット・プロセッサ「80486」の全容」,『日経バイト』, 1989 年 5 月号, PP.257-266

なお，特に以下の図表につきましては，各文献から引用させていただきました.

〔文献 1〕

図 I-2-1，表 I-2-2，図 I-2-19，図 I-2-20，表 I-2-4，図 I-2-23，図 I-2-24，図 I-2-25，図 I-2-26，表 I-2-5，表 I-2-6，図 I-2-28，図 I-2-29

〔文献 4〕

図 E-1，図 E-2，図 E-3，図 E-4，表 E-1

■編集協力

藤田洋史

八谷祥一

川名章史

■表紙写真

前川建二

●本書の内容に関するご質問は、小社第三書籍編集部まで、かならず封書（返信用切手同封のこと）にてお願いいたします。
電話によるお問い合わせには、応じられません。
なお、本書の範囲を越える質問に関しては、お答えできない場合もあります。
●落丁・乱丁本は、送料当社負担にてお取り替えいたします。
お手数ですが、小社営業部までご返送ください。

改訂3版 FM TOWNSテクニカルデータブック

1994年5月1日　初版発行

著　者　千葉 憲昭（ちば のりあき）
発行人　宮崎 秀規
編集人　松本 剛
発行所　**株式会社アスキー**
〒151-24　東京都渋谷区代々木4丁目33番10号
振　替　東京4-161144
大代表（03）5351-8111
出版営業部（03）5351-8194（ダイヤルイン）
第三書籍編集部（03）5351-8184（ダイヤルイン）

編集協力　株式会社スタッフプロモーション
制　作　株式会社ドキュメントシステム
印　刷　図書印刷株式会社

編　集　松本 剛／磯辺 加代子

ISBN4-7561-0467-3　　Printed in Japan

◐11832

改訂3版 FM TOWNS テクニカルデータブック

対応機種

FMTOWNS 1
FMTOWNS 2
FMTOWNS 1F
FMTOWNS 2F
FMTOWNS 1H
FMTOWNS 2H
FMTOWNS 10F
FMTOWNS 20F
FMTOWNS 40H
FMTOWNS 80H
FMTOWNS II UX10
FMTOWNS II UX20
FMTOWNS II UX40
FMTOWNS II CX10
FMTOWNS II CX20
FMTOWNS II CX40
FMTOWNS II CX100
FMTOWNS II UG10
FMTOWNS II UG20
FMTOWNS II UG40
FMTOWNS II UG80
FMTOWNS II HG20
FMTOWNS II HG40
FMTOWNS II HG100
FMTOWNS II HR20
FMTOWNS II HR100
FMTOWNS II HR200
FMTOWNS II UR20
FMTOWNS II UR40
FMTOWNS II UR80
FMTOWNS II ME20
FMTOWNS II ME170
FMTOWNS II MA20
FMTOWNS II MA170
FMTOWNS II MA170W
FMTOWNS II MA340
FMTOWNS II MA340W
FMTOWNS II MX20
FMTOWNS II MX170
FMTOWNS II MX170W
FMTOWNS II MX340
FMTOWNS II MX340W
FMTOWNS II MF20
FMTOWNS II MF170W
FMTOWNS II Fresh

9784756104670

1913055085009

ISBN4-7561-0467-3

C3055 P8500E

定価8,500円(本体8,252円)

改訂3版

FM TOWNS

テクニカルデータブック

千葉憲昭 著

ASCII